Titre: Title:	Techniques de routage pseudo-aléatoire pour une application micro-électronique
Auteur: Author:	Étienne Lepercq
Date:	2012
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Lepercq, É. (2012). Techniques de routage pseudo-aléatoire pour une application micro-électronique [Ph.D. thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1006/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1006/
Directeurs de recherche: Advisors:	Yvon Savaria, & Yves Blaquière
Programme: Program:	Génie électrique

UNIVERSITÉ DE MONTRÉAL

TECHNIQUES DE ROUTAGE PSEUDO-ALÉATOIRE POUR UNE APPLICATION MICRO-ÉLECTRONIQUE

ÉTIENNE LEPERCQ

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION

DU DIPLÔME DE PHILOSOPHIAE DOCTOR

(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

TECHNIQUES DE ROUTAGE PSEUDO-ALÉATOIRE

POUR UNE APPLICATION MICRO-ÉLECTRONIQUE

présentée par : LEPERCQ Étienne

en vue de l'obtention du diplôme de : Philosophiae Doctor

a été dûment acceptée par le jury d'examen constitué de :

M. DAVID Jean-Pierre, Ph.D., président

M. SAVARIA Yvon, Ph.D., membre et directeur de recherche

M. BLAQUIÈRE Yves, Ph.D., membre et codirecteur de recherche

M. BOIS Guy, Ph.D., membre

M. HABIB Mehrez, Ph.D., membre

Remerciements

Je remercie d'abord et avant tout mes
professeurs et directeurs de thèse :
Yvon Savaria et Yves Blaquière.
Cette thèse et ces recherches n'étaient
pas possible sans vous.

Je remercie l'équipe *DreamWafer*,
en particulier Yan B.B, Nicolas L.M.,
Walder A., Olivier V. : on a énormément
travaillé sur ce projet, c'était
passionnant et « ça valait la peine ».

Je remercie les partenaires industriels
du projet : vos exigences m'ont fait me
dépasser plus d'une fois, pour des
choses qui bien souvent ne se
retrouvent pas dans cette thèse.
J'ai appris, et pas seulement
sur le plan technique.

Je remercie les fonds de recherche
MITACS ainsi que Gestion TechnoCap Inc.,
qui m'ont supportés financièrement
tout au long de mon doctorat.
Grâce à vous, je n'ai fait
aucun compromis sur mes recherches
et suis resté pleinement concentré.

Je remercie ma famille : c'est plus que merci,
ce chemin je l'ai fait avec vous, avec toutes
vos forces, votre amour, vos espoirs aussi.

Je remercie ma femme Emanuelle F.,
qui m'a tant supporté (surtout sur la fin !).

Je t'aime.

Résumé

La problématique de routage est très actuelle. On en trouve des applications dans les *GPS*, les prévisions de trafic routier, mais aussi pour le prototypage sur *FPGA*, la fabrication de puces électroniques ou le trafic *TCP/IP* sur Internet. On trouve des publications sur le sujet depuis plusieurs dizaines d'années, mais on observe actuellement une recrudescence confirmant l'actualité, l'importance et la complexité de ce problème.

Cette thèse concerne le routage et ses ressources pour une application dans un nouveau type de système micro-électronique, nommé le *WaferBoard*™. Son noyau consiste en un circuit électronique intégré à l'échelle d'une tranche de silicium (*wafer*). Peu d'applications commerciales de la micro-électronique ont exploité ce niveau d'intégration. Ce système de prototypage rapide vise à réduire d'un ou deux ordres de grandeur le temps de développement de systèmes électroniques. Il nécessite un ensemble d'outils logiciel de support, dont un outil de routage très rapide, capable de produire des solutions valables en des temps de l'ordre de la minute, et de certaines fonctionnalités spécifiques, l'équilibrage de délai ou le reroutage à la volée, au sein d'une netlist déjà routée.

La problématique de routage pour cette application peut être imagée comme suit. Étant donné un réseau routier régulier (les routes d'Amériques du Nord en version cartésienne par exemple) et 100,000 voitures au départ lundi à 8h a.m. dans tout le pays avec des sources et destinations très variées ; calculer les chemins pour toutes les voitures de telle sorte qu'aucune ne prenne la même route dans la journée. Il est 7h59 a.m, vous avez 1 minute, et des ponts sont inaccessibles pour travaux, en voici la liste.

Cet exemple simpliste donne une idée des ordres de grandeurs de la problématique de routage que l'on cherche à résoudre pour cette application. Un algorithme de routage prend en paramètres deux structures de données : un graphe (ou réseau d'interconnexions) constitué de nœuds (sommets) et d'arcs [1], et une *netlist* [2], liste de nœuds électriques dont les points de départ et d'arrivée sont positionnés géographiquement. Ainsi, au lieu de voitures, il s'agit de router des signaux électriques dont les points de départ et d'arrivée sont dictés par la position des broches des composants placés sur le système de prototypage. Un réseau régulier maillé mufti-dimensionnel (plus généralement appelé « réseau d'interconnexions ») sert de réseau routier dont certaines routes sont défectueuses, des ponts inaccessibles. En effet, le réseau d'interconnexions est un circuit électronique intégré à l'échelle d'une tranche de silicium complète, ce qui

1. Un arc relie deux sommets du graphe
2. Dans ce contexte, un *netlist* réfère à une liste d'interconnexions entre composants

implique la présence de défectuosités au sein de chaque circuit fabriqué. Contrairement aux circuits électroniques classiques, où chacun est testé et les défectueux écartés, une intégration à l'échelle de la tranche demande de fortes redondances au sein du circuit pour minimiser le taux de rejets. Pour l'application du *WaferBoard*, un certain nombre d'éléments du réseau d'interconnexions seront fort probablement défectueux sur chaque circuit produit ; l'algorithme de routage se doit de prendre en compte ces éléments très particuliers. Cette contrainte ne se retrouve pas dans les applications plus classiques des routeurs que l'on retrouve dans les *PCB*, circuits *FPGA* ou circuits *VLSI*. D'autres contraintes s'appliquent à ce projet particulier : la latence induite par la technologie est environ un ordre de grandeur plus importante que celle dans les circuits sur *PCB*, ce qui impose un routage orienté vers sa réduction. Cette contrainte explique en partie l'architecture du réseau utilisé. Au sein du *WaferIC*, la distance la plus courte n'est pas forcément celle qui offre la latence la plus petite. Cette propriété du réseau complexifie quelque peu le routage. L'équilibrage des délais au sein d'un groupe de *net* de taille arbitraire est une fonctionnalité nécessaire de l'algorithme de routage, dont la difficulté de l'approche est décuplée par les temps de calcul visés. En effet, l'intérêt du *WaferBoard* est la rapidité pour l'utilisateur de tester un circuit : le temps de mise en œuvre est extrêmement court, et estimé à quelques minutes seulement. Les temps de calculs pour le routage devront donc rester inférieurs à 10 minutes environ, soit l'ordre de grandeur acceptable par l'utilisateur.

Pour situer les travaux de cette thèse, une revue de littérature approfondie du domaine des routeurs pour les outils de conception de systèmes électroniques est réalisée dans cette thèse, pour en tirer une classification des différentes familles d'algorithmes de routage connues. Il apparait que la majorité des approches aujourd'hui classiques appliquées aux circuits *FPGA* et *VLSI* se focalisent sur la qualité du routage final au détriment des temps de calcul qui sont donc généralement très longs. Un mouvement récent apparait dans la littérature, qui vise la construction d'algorithmes qui routent plus rapidement sans trop perdre en qualité. Les recherches présentées dans ce document s'associent à ce mouvement.

Parmi mes différentes contributions, une technique de routage proposée est caractérisée par le calcul de permutations des arcs utilisés par chaque route. Ainsi, chaque permutation correspond à une nouvelle solution pour relier deux points du circuit, équivalente en terme de délai, et possiblement non conflictuelle. Cette technique utilisée seule ne suffit pas, mais elle est compatible avec des approches séquentielles utilisées dans la littérature, comme l'algorithme A^*. Il est probable que la littérature soit passée à côté de ces techniques en raison de l'intérêt général pour des techniques de routage permettant de router avec succès des netlist toujours plus denses. Cependant des réductions de temps de calcul intéressantes peuvent être obtenues avec la technique proposée. Une autre contribution concerne un algorithme pour calculer le plus court chemin dans un graphe mufti-dimensionnel, avec une complexité de calcul linéaire avec la distance à parcourir, indépendante de la taille du graphe. Cette complexité est donc très largement inférieure aux routeurs de type labyrinthe tel le A^*, qui ont globalement une complexité quadratique en fonction du nombre total de nœuds du graphe. La démonstration mathématique de cette approche est proposée au chapitre 2.2.6 avec les techniques

décrites dans ce paragraphe.

En plus de ces techniques de routage qui considèrent chaque *net* indépendant, une autre contribution ajoute un algorithme d'équilibrage des délais. Fonctionnalité majeure de l'algorithme de routage pour le WaferBoard, il vise à assurer que la différence de temps de propagation au sein d'un groupe de routes soit inférieur à une borne spécifiée par l'utilisateur. Bien sûr, il faut également s'assurer que les routes calculées ne soient pas en conflit avec d'autres. L'équilibrage des délais est donc une contrainte supplémentaire, et même prioritaire, au routage. De plus, l'équilibrage n'est pas forcément possible en une itération, il est nécessaire de construire un algorithme qui relaxe progressivement les délais maximums autorisés, pour assurer le fonctionnement du circuit. La latence augmente, mais l'équilibrage est respecté. L'algorithme proposé est construit selon deux axes : le premier est une extension d'un algorithme publié récemment dans le domaine des *FPGA* appelé *Routing Cost Valley* (*RCV*). Celui-ci est robuste, non-aléatoire et fondé sur le A*. *RCV* tient compte lors du routage de contraintes de délais minimum et maximum pour un *net* (borne), mais ne gère pas les contraintes associées à des groupes. L'extension proposée le permet par l'adjonction de deux routines de gestion des groupes de *net*. Les contraintes associées à l'équilibrage des délais augmentent fortement les temps de calcul, observation que l'on retrouve dans la littérature. Un algorithme d'accélération est ainsi proposé, algorithme qui fusionne avec celui basé sur *RCV*. Cette contribution utilise une propriété des réseaux d'interconnexion *CMOS*, qui sont l'allongement du délai d'une route par l'utilisation de plusieurs segments, au lieu d'un seul. Les propriétés de cette technique sont analysés au chapitre 4.

Le routeur pour ce projet traite des *netlist* similaires à celles pour *PCB*, et utilise un réseau d'interconnexions particulier. Or, bien qu'il existe plusieurs *netlist* admises comme bancs d'essais pour comparer les algorithmes sur *FPGA* et *VLSI*, il n'en existe pas pour les *PCB*. Des ensembles de données existent comme celui qui était à la base d'une compétition sur les outils de routage, mais malheureusement, ces données ne sont pas publiques ; la recherche académique sur les routeurs *PCB* n'est donc pratiquement plus active, et ce depuis plus de 10 ans. Les *netlist* placés sont des informations confidentielles pour l'industrie. Plusieurs publications existent sur les générateurs de *netlist* synthétiques au sein des communautés travaillant sur *FPGA* et circuits *VLSI* mais aucune pour les *netlist* de *PCB*. Les *netlist* synthétiques ont pour avantage d'être modulables à souhait selon différentes métriques (densité, complexité d'interconnexions, qualité du placement par exemple). Idéalement, il est possible de moduler ces métriques indépendamment les unes des autres, ce qui facilite la caractérisation des algorithmes, afin de mieux visualiser leurs forces et faiblesses. Comme contribution, j'ai réalisé le premier générateur de *netlist* synthétiques pour *PCB* publié. Dans le cadre de cette thèse, il est démontré qu'une métrique, la loi de Rent, couramment utilisée pour représenter les *netlist* sur *FPGA* et *VLSI*, est inapplicable aux *PCB* modernes. Ainsi, un modèle novateur fondé sur deux métriques importantes, la distribution du degré des *net* et celle de la longueur des *net*, a été développé pour les besoins du projet. Ce modèle a conduit à la construction d'un générateur qui permettra à des chercheurs d'obtenir facilement des *netlist* pour *PCB*, et ainsi établir les bases d'un ensemble de bancs d'essais réalistes pour *PCB*. Ces résultats sont présentés au chapitre 4.4.

Pour terminer, une contribution de mon travail est la réalisation d'un outil fonctionnel pour le WaferBoard : l'algorithme, les structures de données et même l'interface graphique sont intégrés au sein du logiciel. Bien que ne constituant pas une contribution scientifique du même ordre, il s'agit d'une réalisation majeure en terme pratique qui a contribué à l'avancement du projet dans sa globalité.

Abstract

The routing problem is very actual. Applications are found in *GPS*, road traffic forecast, but also for prototyping on *FPGA*, or *TCP/IP* traffic on the Internet. Publications on the subject have existed for several decades, but new publications keep appearing, confirming the importance and complexity of the problem.

This thesis deals with routing and the resources it requires for a new category of micro-electronic applications, called the *WaferBoard*. It is an electronic circuit integrated at the wafer scale. Few commercial applications of micro-electronics have exploited this level of integration. This rapid prototyping system aims at reducing by one or two orders of magnitude the development time of digital circuits. It requires a very fast routing tool, capable of producing viable solutions in a few minutes, with dedicated functionality such as balancing delays and rerouting on the fly parts of a netlist.

The routing problem for this application can be pictured as follows. Given a regular road network of the size of north america, if 100.000 cars were to start Monday 8 a.m. across the continent with a wide variety of sources and destinations ; the challenge is to compute paths for all cars so none of them take the same route that day. It is 7 :59 am, you have 1 minute, and some bridges are under road work : here is the list.

This simplistic example gives an idea of the orders of magnitude of the problem that need to be solved for this application. A routing algorithm takes as input : a graph (or interconnection network) made of nodes and edges, and a netlst, a list of electrical nodes with starting and ending points physically placed. Therefore, instead of cars, the problem consists of routing electrical signals with points of departure and arrival dictated by the pin position of components placed on the prototyping system. A regular, multi-dimensional mesh (also called "interconnection network") serves as a road network, which contains defective roads and inaccessible bridges. Indeed, the interconnection network is an electronic circuit integrated across a full wafer, implying the presence of defects within each manufactured circuit. Unlike conventional electronic circuits, where each is tested and defective ones are set apart, wafer scale integrated applications require lots of redundancy in the circuit to minimize the rejection rate. In the WaferBoard, a number of elements of the interconnection network will be defective in each circuit ; the routing algorithm must take into account these very specific elements. This constraint is not found in the classic applications of routers found in *PCB*, *FPGA* or *VLSI* circuits. Other restrictions apply to this particular project : the latency induced by the technology is about one order of magnitude greater than that in the circuits of *PCB*s, which requires a routing oriented towards computation time

reduction. This constraint partly explains the network architecture used. Within the *WaferIC*, the shortest distance is not necessarily the one that offers the smallest latency. This property of the network complexifies the routing problem. Balancing delays within a group of arbitrary size nets is a necessary feature of the routing algorithm, and the difficulty is amplified by the computation time limit. Indeed, the interest of the application is to reduce the time for a user to test a circuit : the time of setup is extremely short, and estimated at a few minutes only. The computation time for the routing should ideally stay under 10 minutes, which is acceptable to the user.

A complete literature review of routers for the electronic technologies is performed in this thesis. It allows us to classify the different families of known routing algorithms. It appears that the conventional approaches now used for *VLSI* circuits and *FPGA*, which include the vast majority of publications today, focuses on the quality of the final routing at the expense of computation time that are generally very long. A recent trend appears in the literature, which involves algorithms trying to route faster without losing quality. Researches presented in this document are associated with this movement.

Among various contributions, a proposed routing technique is characterized by calculating permutations of edges used by each route. Thus, each permutation correspond to a new solution to connect two points of the circuit, equivalent in terms of delay, and possibly non-conflicting. Alone, this technique is not sufficient for routing, but it is compatible with sequential approaches used in the literature, such as the A * algorithm. It is likely that these techniques were not developed in the literature, because of general interest for pushing the limits of maximum density. However, interesting reductions of computation time can be obtained with the proposed techniques. Another contribution concerns an algorithm to compute the shortest path in a multi-dimensional graph that has a linear complexity with the distance to travel, independent of the size of the graph. This complexity is far less than that of labyrinth algorithms, such as the A *, which have an overall complexity that depends on a quadratic function of the total number of nodes in the graph. A mathematical demonstration of this approach is proposed in Chapter 2.2.6, along with the other techniques briefly described in this paragraph.

In addition to simple routing techniques that consider each net independently from others, a delay balancing algorithm is proposed. As a major feature of the routing algorithm for the WaferBoard, it seeks to ensure that the difference in propagation delays in a group of nets is less than some user-specified constraint. Of course, it must also ensure that routed nets are not in conflict with others. Delay-balancing is an additional constraint for routing. Moreover, the balance is not necessarily possible in one iteration, it is required to build an algorithm that gradually relaxes the maximum latency allowed, to ensure the correct operation of the circuit. While latency increases, the balance is respected. The proposed algorithm is built on two foundations : the first one is an extension of an algorithm published recently in the field of *FPGA*, called the *Routing Cost Valley* (*RCV*). It is robust and based on the A *. *RCV* takes into account minimum and maximum delay constraints for a net, but does not handle constraints associated to groups of nets. The proposed extension allows it by the addition of two functions dedicated to group constraints. Constraints associated

with delay balancing greatly increases computation times, an observation found in the literature. An acceleration algorithm is proposed, which merges with *RCV*. It uses a feature of *CMOS* interconnection networks, which is a longer delay with the use of multiple edges, instead of one. The properties of this technique are analyzed in chapter V.

The router for this project deals with netlist similar to those for *PCBs*, and uses a specific interconnection network. However, although there are several netlists generally admitted as benchmarks for comparing algorithms on *FPGAs* and *VLSI*, none exist for *PCBs*. In absence of generally accepted benchmarks, the academic research on *PCB* routers is therefore almost inactive, and this has been so for over 10 years. Placed netlist are confidential information for the industry. However, literature on synthetic netlist generators exists in communities working on *FPGAs* and *VLSI ASICs*, but none are avalable for *PCBs*. Synthetic netlist have the advantage of being adjustable at will according to different metrics (density, complexity of interconnections, placement quality for example). Ideally, it is possible to modulate these metrics independently of each other, which facilitates the characterization of routing algorithms, in order to better visualize their strengths and weaknesses. This thesis reports the first published synthetic netlist generator for *PCBs*. As part of this thesis, we demonstrate that a well-known metric, the Rent's rule, commonly used to represent *FPGA* and *VLSI* netlists, is inapplicable to modern *PCBs*. Thus, an original model based on two important metrics, the distribution of the net degree and the net length distribution, has been developed for the needs of the project. This model has led to the construction of a generator that will allow researchers to easily obtain netlists for *PCBs*, and establish the basis of a set of realistic benchmarks for *PCBs*. These results are presented in chapter 4.4.

Finally, a practical contribution of this work is the implementation of a functional tool to support that WaferBoard : algorithms, data structures and even a GUI were integrated within a complete software. While this is not a pure scientific contribution, it is a major practical contribution to enginnering that supported the demonstration that tools for supporting the WaferBoard technology are feasible.

Table des matières

Liste des tableaux

Table des figures

Liste des symboles

ASIC En anglais Application Specific Integration Circuit. Un circuit intégré (micro-électronique) dédié à une application particulière, réalisé avec l'aide de portes logiques assemblées, contrairement aux circuits dits full-custom. En général, il regroupe un grand nombre de fonctionnalités uniques et/ou sur mesure.

BSG En anglais Bounded Slice-surface Grid. Algorithme capable de réaliser un placement en fonction de contraintes qui caractérisent une fonction coût.

CAD En anglais Computed Aided Design (conception assistée par ordinateur). Comprend l'ensemble des logiciels et des techniques de modélisation géométrique permettant de concevoir, de tester virtuellement - à l'aide d'un ordinateur et des techniques de simulation numérique - et de réaliser des produits manufacturés et les outils pour les fabriquer. Source Wikipedia France.

DCLC En anglais Delay-constraint least cost problem (Problème du plus petit coût avec contrainte de délai). Problème géréral théorisé par les chercheurs dans le domaine des télécommunications.

FLUTE En anglais Fast Lookup Table Based Rectilinear. Algorithme capable de résoudre le problème d'arbre de Steiner, c'est-à-dire un problème d'optimisation combinatoire proche de l'arbre couvrant minimal.

FPGA En anglais Field Programmable Gate Arrays. Un circuit intégré logique qui peut être reprogrammé après sa fabrication. Source Wikipedia France.

FPIC En anglais Field Programmable Integrated Circuit. Un circuit reprogrammable, en particulier au niveau de ses entrées/sorties. Ils sont surtout utilisés pour des applications qui nécessitent de la flexibilité dans les interconnections, tout en offrant des vitesses de transfert importantes.

PCB En anglais Printed Circuit Board. Support, en général une plaque, permettant de relier électriquement un ensemble de composants électroniques entre eux, dans le but de réaliser un circuit électronique complexe. Source Wikipedia France.

QoS En anglais Quality of service. Capacité à véhiculer dans de bonnes conditions un type de trafic donné, en termes de disponibilité, débit, délais de transmission, gigue, taux de perte de paquets... Source Wikipedia France.

RCV En anglais Routing Cost Valley. Algorithme de routage capable de prendre en compte une contrainte de délai minimal et maximal pour une route donnée.

TSV En anglais Through Silicon Vias. Une technique de fabrication qui permet de percer des trous dans un wafer de silicium. En géréral, le perçage est réalisé par le dessous du wafer, pour atteindre une couche de métalisation. Un perçage complet est toutefois possible.

VLSI En anglais Very-Large-Scale Integration. Technologie de circuit intégré (CI) dont la densité d'intégration permet de supporter plus de 100 000 composants électroniques sur une même puce. Source Wikipedia France.

VPR En anglais Versatile Place and Route. Outil capable de réaliser le placement et le routage global et détaillé pour FPGA.

CMOS En anglais Complementary Metal Oxide Semiconductor. Technologie de fabrication de composants électroniques et, par extension, l'ensemble des composants fabriqués selon cette technologie. À l'instar de la famille Transistor-Transistor logic (TTL), ces composants sont en majeure partie des portes logiques (NAND, NOR, etc.) mais peuvent être aussi utilisés comme résistance variable. Source Wikipedia France.

CPU En anglais Central Processing Unit. Le composant de l'ordinateur qui exécute les programmes informatiques. Avec la mémoire notamment, c'est l'un des composants qui existent depuis les premiers ordinateurs et qui sont présents dans tous les ordinateurs. Un processeur construit en un seul circuit intégré est un microprocesseur. Source Wikipedia France.

GPS En anglais Global Positionning System. Un système de géolocalisation fonctionnant au niveau mondial. En 2011, il est avec GLONASS, un système de positionnement par satellites entièrement opérationnel et accessible au grand public. Source Wikipedia France.

ISCAS En anglais International Symposium on Circuits and Systems.

ISCAS En anglais International Symposium on Circuits and Systems.

ISPD En anglais International Symposium On Physical Design. Conférence sur le design physique, principalement les outils d'aide à la conception.

MCNC En anglais Microelectronics Center of North Carolina. Dans cette thèse, réfère aux bancs d'essais de circuits VLSI pour le placement et le routage.

PL Programmation Linéaire. Un problème d'optimisation linéaire est un problème d'optimisation dans lequel on minimise une fonction linéaire sur un polyèdre convexe. Source Wikipedia France.

PLE Programmation Linéaire Entière. Problème identique à un problème de Programmation Linéaire, mais dans lequel l'ensemble des variables du problème prennent des valeurs entières.

R and R En anglais Rip-up and Reroute. Technique de routage utilisée par les algorithmes séquentiels, qui consiste à sélectionner une liste de routes à enlever, puis router à nouveau avec un nouvel ordre de priorité.

RAM En anglais Random Access Memory. Mémoire dont les informations sont perdues lors de l'extinction du système.

RSMT En anglais Rectilinear Steiner Minimum Tree. Arbre rectiligne de Steiner minimal.

TCP/IP En anglais Transmission Control Protocol, Internet Protocol. La suite TCP/IP est l'ensemble des protocoles utilisés pour le transfert des données sur Internet.
Elle est souvent appelée TCP/IP, d'après le nom de deux de ses protocoles : TCP (Transmission Control Protocol) et IP (Internet Protocol), qui ont été les premiers à être définis. Source Wikipedia France.

Introduction

Les recherches présentées dans cette thèse sont nées des besoins et contraintes particulières d'une nouvelle application micro-électronique : le *WaferBoard*™. Cette application fut développée dans le cadre d'un projet de recherche appelé *DreamWafer*™ dans lequel plusieurs partenaires industriels sont également impliqués. Les principales universités participantes sont l'École Polytechnique de Montréal, l'Université du Québec à Montréal, l'Université du Québec en Outaouais. Le principal partenaire industriel est Gestion TechnoCap Inc.

Le marché de l'électronique oblige, pour rester compétitif, à augmenter fortement la complexité et le nombre de composants intégrés dans un même circuit. En conséquence, la complexité de mise au point et le temps de développement des *PCB* (*Printed Circuit Board*) augmentent fortement. L'augmentation de la densité des boitiers des composants entraine la mise au point de nouvelles technologies pour les supporter. C'est dans ce contexte que le projet mentionné vise la réalisation d'une nouvelle plateforme de prototypage rapide de systèmes électroniques. Son objectif est de permettre de réduire le cycle de développement de systèmes complexes de plusieurs mois à quelques jours, sinon quelques minutes, tout en réduisant le coût de fabrication du prototype. Une solution existante est l'utilisation des *FPIC* (*Field Programable Integrated Circuit*) [10] permettant de programmer quelques milliers de connexions sur un *PCB* classique. Plusieurs limitations existent avec cette technologie (en dehors de son coût) notamment la nécessité de produire un *PCB* pour chaque prototype avec les empreintes exactes de chaque composant utilisé, ou la nécessité d'étager plusieurs *FPIC* (dans un réseau de type Benes [11]) pour des systèmes complexes. Ces contraintes supplémentaires réduisent fortement le taux d'intégration sur une même carte. Le projet *DreamWafer* enlève ces inconvénients tout en accélérant la mise au point du matériel et du logiciel prototypé. Le cycle de développement est assuré par un logiciel dont les principaux aspects sont décrits plus loin. Pour alléger cette section, une revue de littérature complète des algorithmes de routage est faite au chapitre 2. Les contraintes et spécificités du projet qui touchent l'algorithme de routage sont résumés ci-après :

- Taille importante du réseau, comparée aux autres technologies comme les *VLSI* (*Very Large Scale Integration*) et *FPGA* (*Field Programmable Gate Arrays*). Le tableau 0.1 présente une comparaison des tailles respectives des technologies ;
- Architecture du réseau différente de celles utilisées sur circuits *FPGA*, *VLSI* ou *PCB* ;
- Absence d'algorithme de placement intelligent (le placement est réalisé par l'utilisateur, pour un prototype qui

ne durera pas) tel qu'utilisé pour les *FPGA* et *ASIC* ;

- Ressources de routage amputées par les défectuosités lors de la fabrication ;
- Contraintes fortes de temps de routage, de l'ordre quelques minutes ;
- Délai de propagation non monotone en fonction de la distance ;
- Le routeur prend comme entrée des *netlist* similaires à celles qui décrivent des systèmes réalisés à l'aide de circuits imprimés *(PCB*), difficiles à obtenir de la part d'industriels pour vérifier et étalonner les algorithmes implémentés.

Il est important pour la bonne compréhension de la portée et du contexte des recherches de présenter la plateforme matérielle qui sert de support à ces recherches. Une présentation en est donc faite dans cette introduction. Une analyse des points communs et des différences majeures avec d'autres technologies est proposée. Enfin, un aspect plus logiciel, les interactions de l'algorithme de routage au sein de l'outil complet de développement prévu, *WaferConnect*, est détaillé. Ainsi, cette introduction mettra en lumière le contexte des recherches, les principales contraintes matérielles en contraste avec d'autres domaines tels que les *FPGA*, les *PCB* ou encore les routeurs *VLSI*, ainsi que les contraintes côté logiciel.

Pour appréhender la complexité du *WaferBoard*, il convient de le décrire sous la forme de subdivisions en modules distincts (figure 0.0.1). Le *WaferBoard* se présente sous la forme d'un ordinateur portable en plus épais (environ $20\,\text{cm} \times 15\,\text{cm} \times 10\,\text{cm}$). Il s'ouvre en deux comme montré sur la figure 0.0.1, avec d'un côté le couvercle et de l'autre une surface active appelée *WaferIC*. Il s'agit d'un circuit de la taille d'une tranche complète de silicium, contrairement aux circuits classiques sur *FPGA* et *VLSI*, qui occupent au plus quelques centimètres carrés. Dans le cadre du projet, le circuit qui sert de support à toute la technologie est une puce intégrée à l'échelle d'une tranche (on parle de *Wafer-Scale*) ce qui impose des contraintes uniques, notamment sur l'algorithme de routage qui nous intéresse ici tout particulièrement. Les composants sont déposés par l'utilisateur (sans soudure) sur le *WaferIC* en fonction de ses besoins, tel que schématisé sur la figure 0.0.2. Les billes ou broches doivent être non-traversantes pour être acceptées par la plateforme. Il est à noter que la très large majorité des systèmes électroniques aujourd'hui utilisent des boîtiers dont les broches ne traversent pas le *PCB*. Un film anisotropique appelé film Z (*Z-axis film*) constitué de centaines de millions de minuscules fils de cuivres verticaux garantis sans court-circuits est placé sur la surface du *wafer*. Ce film sert d'interface entre les broches et le *WaferIC* proprement dit. Ce film est élastique (en particulier sur l'axe vertical), ce qui permet d'avoir un bon retour en position de repos lorsque le couvercle est ouvert - la pression étant alors relâchée.

Sous le *WaferIC*, un ensemble de 21 blocs d'alimentations très particuliers appelés *PowerBlock* sont connectés au *WaferIC* par le dessous. Des « trous » appelés *TSV* (*Through Silicon Via*s) dans le *WaferIC* sont réalisés par un flux ionique qui traverse l'épaisseur du substrat jusqu'à rencontrer le cuivre d'une piste de la première couche de métal. Chaque bloc d'alimentation est connecté au *WaferIC* au travers de billes soudées sous les *TSV*, ce qui fournit les masses, deux rails d'alimentation (1.8 V et 3.3 V) et 5 fils pour la configuration du *WaferIC*. En plus de l'alimentation,

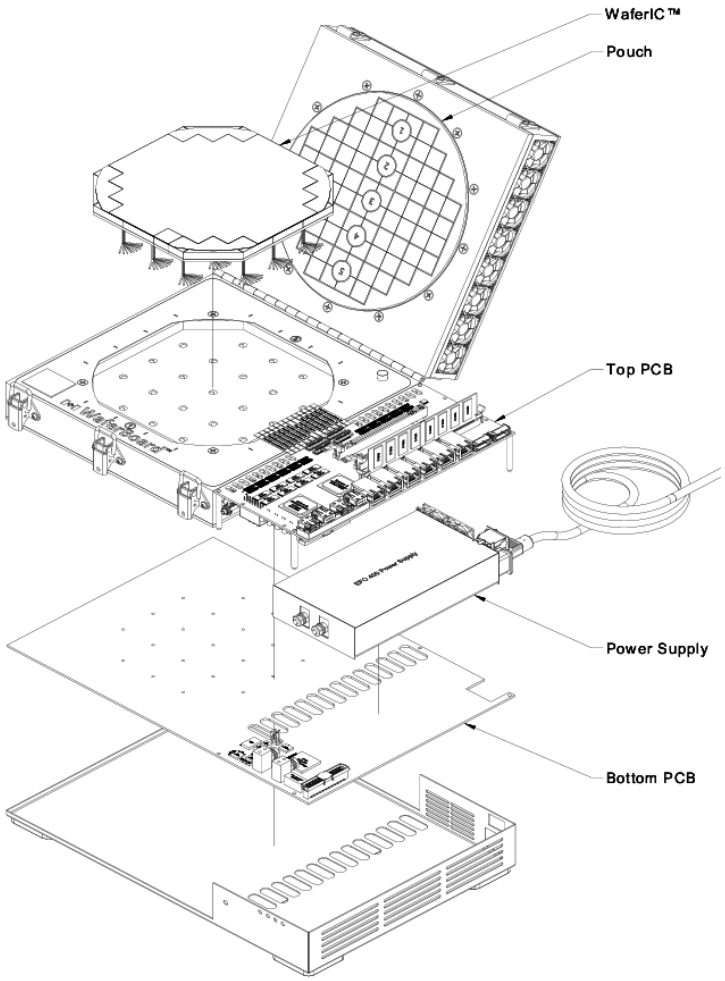


FIGURE 0.0.1: Modèle tridimensionnel du système physique dans son ensemble : le *WaferBoard*^{TM}. Crédit TechnoCap Inc. *DreamWafer* Division. Reproduit avec permission.

ces modules réalisent le lien de communication nécessaire entre le *Bottom-PCB* décrit plus loin, et les *WaferIC*. Un *PCB* appelé *Top-PCB* intègre des connexions vers un ordinateur, des *FPGA* et connecteurs externes destinés à compléter la connectique du *WaferBoard*. Les *FPGA* sur cette carte réalisent le pont de communication entre l'ordinateur de support et le *WaferIC*, mais ils sont également dimensionnés pour permettre d'injecter et de capturer des signaux directement au sein du réseau du *WaferIC*. Ces *FPGA* autorisent donc une communication directe avec les composants déposés sur le *WaferIC*. Un deuxième *PCB* appelé *Bottom-PCB*, placé sous le *WaferIC*, possède les connexions nécessaires vers ses signaux de configuration au travers des *PowerBlock*. Le flot de configuration part d'un ordinateur (celui de l'utilisateur), passe au travers d'un *FPGA* sur le *Top-PCB*, un autre sur le *Bottom-PCB*, est interprété par les *PowerBlock* pour finalement arriver au *WaferIC*. En réalité, un *PowerBlock* supporte 4 images de réticule (en pointillés sur la figure 0.0.2) qui sont des réminiscences de la méthode de fabrication des puces électroniques. Le circuit généré sur le *WaferIC* correspond à chaque image de réticule, et est accessible indépendamment.

Une question d'importance qui n'a pas encore été abordée suite à cette rapide présentation du WaferBoard, est de

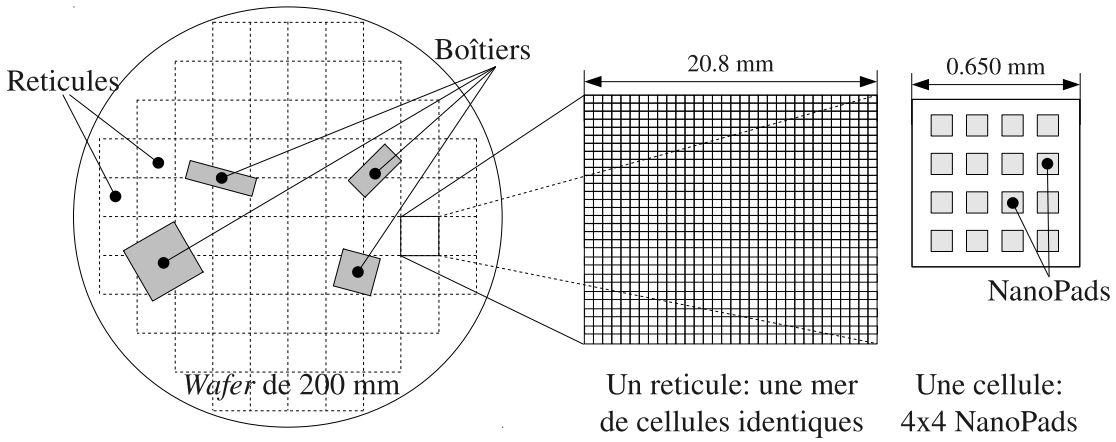


FIGURE 0.0.2: Architecture du *WaferIC*, vue schématique peu détaillée. La mer de cellules permet au circuit d'être répété à l'infini.

comprendre comment les composants placés sur le *WaferIC* peuvent être interconnectés de la même façon que sur un *PCB*. Cet aspect est au cœur des contraintes de l'algorithme de routage. Pour le saisir il est nécessaire d'expliquer quelques détails de l'architecture interne du *WaferIC*.

Celui-ci est un substrat configurable fabriqué par juxtaposition d'images d'un réticule. De la même façon que tout circuit électronique sur silicium, chaque image de réticule correspond à un circuit à part entière. Cependant, les circuits classiques (sans redondance) sont testés sur la tranche directement, les fonctionnels sont conservés (et découpés) et les autres jetés, ou reconditionnés. Au contraire, le *WaferIC* est une seule tranche non découpée ; une opération non conventionnelle supplémentaire est réalisée lors de la fabrication pour « imprimer » des pistes de métal qui relient les circuits de chaque réticule entre eux, appelé *reticle stitching*. Ainsi, le *WaferIC* est donc bien un seul et même circuit de silicium.

Chaque image de réticule est constituée d'une mer de cellules identiques (figure 0.0.2), couvertes à leur surface de 4×4 NanoPads, qui sont autant de points de contact possibles avec des billes de composants posés sur le wafer. Chaque cellule est un carré dont le côté mesure environ 650 μm. L'espacement des NanoPads est petit devant la taille des billes des boîtiers présentement sur le marché, ce qui assure un contact d'un minimum de 2 NanoPads par bille. La cellule constitue la brique de base du circuit, et elle est répétée sur une image complète de réticule (32×32 cellules). Le nombre de NanoPads total est d'environ 1.8 millions, autant d'entrées et sorties qui peuvent être redirigées vers le réseau d'interconnexions interne au *WaferIC*.

C'est ce réseau d'interconnexions qui sert de support à l'algorithme de routage pour réaliser un circuit fonctionnel. Du point de vue matériel, ce réseau est constitué de nœuds et d'arcs. Chaque nœud représente une cellule physique. Les arcs sont les nombreuses pistes métalliques qui relient les cellules entre elles. La figure 0.0.3 présente un schéma de quelques liens pour une cellule (hachurée). Pour chaque cellule, on retrouve les connexions présentes pour un réseau maillé classique, soit les 4 plus proches voisins selon les 4 points cardinaux. À ces connexions sont ajoutées des arcs

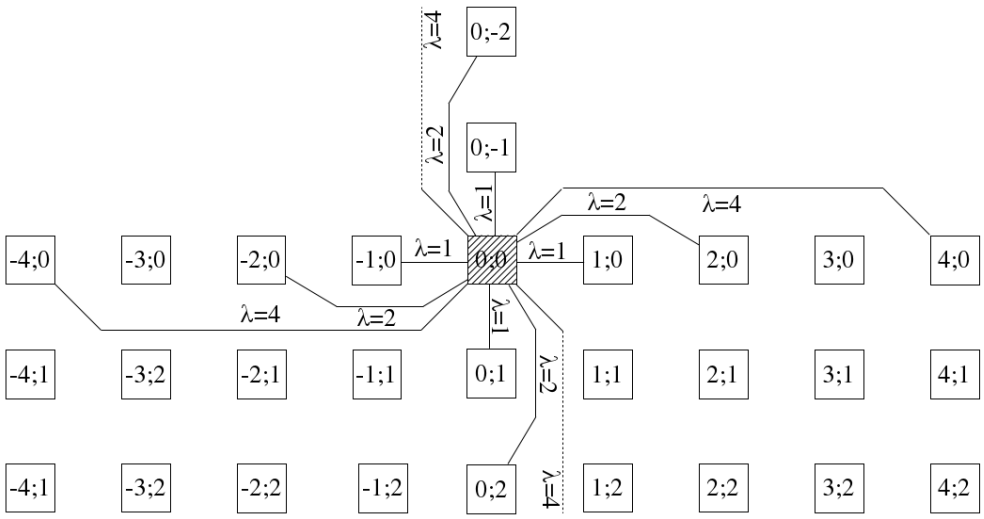


FIGURE 0.0.3: Architecture du réseau d'interconnexions intégré au *WaferIC*. Chaque cellule (représentée par un carré avec une coordonnée) possède des liens (arcs) vers ses 4 plus proches voisines, 4 voisines à distance 2, distance 4, ... jusqu'à 32 dans l'implémentation actuelle. Ainsi, ils forment un réseau multi-dimensionnel. Seuls sont représentés les liens de la cellule 0 ;0 sur quelques distances, mais le patron est répété pour chaque cellule.

connectés aux cellules de distance 2, 4, 8, 16, 32 selon les 4 points cardinaux, soit 6 longueurs différentes. Enfin, chaque arc est unidirectionnel, il existe donc 2 arcs pour chaque paire de cellule source-destination, un dans chaque direction. Au total, le nombre de connexions d'une cellule vers ses voisines directes est de 2 directions $\times$ 4 points $\times$ 6 longueurs $=$ 48 arcs. Un réseau maillé mufti-dimensionnel possède des arcs de longueur $2^n, n \in [0:N]$ et $N \in \mathbb{N}$. Le degré d'un tel réseau réfère à n, le degré de l'arc le plus grand. Spécifier n suffit à caractériser le réseau. Ainsi, le *WaferIC* possède un réseau de degré 5 ($2^5 = 32$). La toute première version, qui a été testée physiquement sur une puce de test fabriquée par TSMC, possédait un réseau de degré 6.

Il est important de noter que contrairement au schéma de la figure 0.0.3, il n'y a physiquement aucun espace entre les cellules : elles sont juxtaposées à la perfection. Étant donné que les cellules font chacune 650 μm, un arc de 32 cellules de long mesure près de 20 mm. Dans un circuit de silicium, les délais d'interconnexions croissent de façon quadratique avec la longueur des fils (du fait de l'accroissement simultané de la résistance et de la capacité parasite des interconnexions dans un circuit intégré). Des répéteurs sont donc insérés une cellule sur deux pour chaque fil, à l'exception des arcs de longueur 1. Grâce à la présence des répéteurs, le délai est ainsi linéairement proportionnel avec la longueur parcourue. Il a été décidé lors de la conception de ne pas utiliser des interconnexions sans répéteur du même type que celles que l'on retrouve dans les *FPGA*, principalement du fait de la grande distance parcourue en moyenne par les signaux dans le *WaferIC*.

Tout ce réseau serait inutile si l'on ne pouvait configurer les connexions entre les nœuds (c'est-à-dire entre les cellules). Réaliser des connexions physiques entre des broches de composants déposés à la surface du *WaferIC* impose de faire circuler des informations (trains de bits) de cellules en cellules, jusqu'à destination. Pour ce faire, un *crossbar*

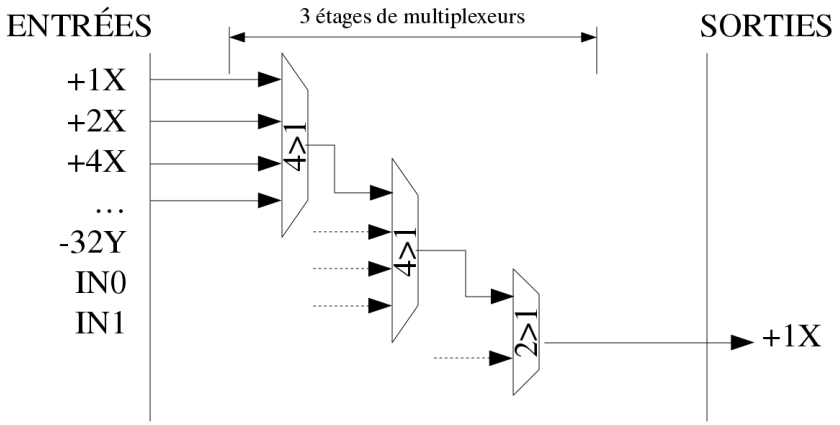


FIGURE 0.0.4: Schéma de l'architecture interne du *crossbar*, implémentation actuelle. Seul un groupe d'entrées vers une sortie sont représentés, les autres sont identiques. Chaque entrée traverse 2 multiplexeurs 4 vers 1 et 1 multiplexeur 2 vers 1.

est intégré à chaque cellule. L'architecture actuelle est constituée d'un *crossbar* de 26 entrées $\times$ 28 sorties ($4 \times 6 = 24$ pour les liens entrants et sortant, plus 2 entrées connectées aux sorties des NanoPads et 4 sorties spéciales vers les NanoPads qui permettent de supporter des entrées-sorties dynamiques). Il est possible de configurer quelle entrée est redirigée vers chaque sortie (figure 0.0.4). Le signal électrique traverse l'équivalent d'un multiplexeur 32 vers 1. La sélection se fait par la configuration d'un registre de 5 bits pour chaque sortie. Ce *crossbar* occupe une surface importante, environ 25 % de la cellule : il s'agit du plus gros module numérique, la majorité des ressources étant occupées par des modules analogiques (puissance notamment). Les ressources de routage au sein d'une cellule utilisées par le réseau sont de 26 entrées et 26 sorties, en plus des 228 liens venant des cellules voisines, qui la traverse sans passer par le *crossbar*.

Pour avoir une vue plus nette de ce système complexe, la figure 0.0.5 présente un exemple de configuration du *WaferIC* de telle sorte qu'une broche d'un composant puisse communiquer avec une autre, situés à deux places arbitraires à la surface de la tranche. Une bille du composant 1 est un contact avec un NanoPad (au travers du film Z qui sert d'interface). Le buffer d'entrée de ce NanoPad est configuré (niveau de tension, sens). La cellule à laquelle il appartient contient 15 autres NanoPads : tous sont connectés à deux multiplexeur 16 vers 1. Ces deux sorties de multiplexeurs sont dirigées vers deux entrées spécifiques du *crossbar*. Ainsi, il est possible de diriger les signaux de deux NanoPads différents sur une même cellule (soit 2 billes par cellules). Le signal du composant 1 est redirigé vers une cellule en direction de la destination, calculé par l'algorithme de routage. Il faut donc configurer la bonne sortie du *crossbar* pour sélectionner l'entrée correspondante au multiplexeur. Le signal peut franchir *n crossbar* jusqu'à sa destination, qui est la cellule visée. Le *crossbar* de destination est configuré pour diriger le signal vers une des sorties reliée au démultiplexeur, image du multiplexeur pré-cité. Enfin, le buffer de sortie pilote la bille du composant 2 : un signal se propage d'un point à un autre au sein du *WaferIC*, son chemin est configurable et modifiable à souhait.

Le réseau d'interconnexions précédemment décrit partage des similitudes avec ceux d'autres technologies, mais

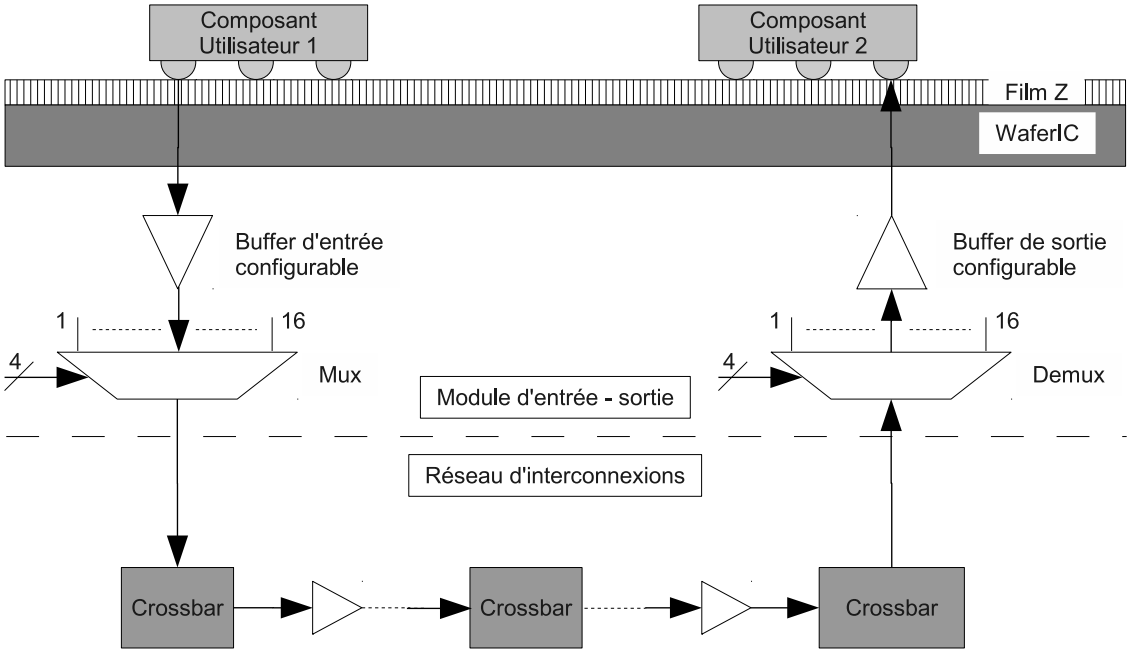


FIGURE 0.0.5: Schéma de principe de configuration et de parcourt d'un signal électrique entre deux broches en contact avec le *WaferIC*. Le signal électrique d'une bille du composant 1 de l'utilisateur se propage dans le réseau d'interconnexions, en passant par les étages d'entrées et de sorties jusqu'à la bille désirée du composant 2.

Tableau 0.1: Comparaison des réseaux d'interconnexions de type Routier, *PCB*, *ASIC*, *FPGA* et multi-dimensionnel (*WaferIC*), selon différentes métriques. Les dimensions sont rapportées en prenant la technologie la plus avancée actuellement publiée.

Métrique	Routier	*PCB*	*ASIC*	*FPGA*	*WaferIC*
Peu dense	✔	✔	✔	✔	✔
Régulier	✖	✔	✔	✔	✔
Taille (# nœuds - # arcs)	3×10^7 - 6×10^7 cf. [3]	6×10^8 - 3×10^9 cf. [4]	$\sim 10^6$ - 6×10^6 cf. [5]	3×10^4 - $\sim 5 \times 10^5$ cf. [6]	9×10^5 - 4×10^6
Délai vs distance est une fonction monotone	✖	✔	✔	✔	✖

possède aussi ses propres caractéristiques et contraintes. Le tableau 0.1 fait une synthèse des similitudes et différences de réseaux d'interconnexions utilisés dans plusieurs technologies, choisies pour leur proximité avec le *WaferIC*.

Tous les réseaux présentés au tableau 0.1 sont, au sens mathématique, de faible densité. Cela signifie au sens commun que chaque nœud est connecté à peu de ses voisins au regard du nombre total de nœuds. Des disparités existent cependant entre les différents réseaux présentés. La connectivité des nœuds sur *PCB* est de 6 ou 10 (4 ou 8 voisins sur un plan, plus deux sur l'axe vertical). Elle est souvent de 6 pour *ASIC* (4 voisins sur un plan, plus deux sur l'axe vertical). La connectivité sur *FPGA* est un peu plus importante, mais varie d'une compagnie à une autre : elle est supérieure à 6, en général inférieure à 20 [12]. Les réseaux routiers sont faiblement connectés en moyenne

(mais sont irréguliers). Le réseau d'interconnexion du *WaferIC* est lui très connecté, avec 24 voisins pour chaque nœuds (sachant que le nombre de fils est doublé, un dans chaque direction, et donc chaque connexion est disponible dans les deux directions). En ce qui concerne les dimensions des graphes équivalents aux réseaux rapportés dans le tableau 0.1, il s'agit de tailles maximales utilisées pour des systèmes de très grande complexité, ou des technologies de prototypage : les dimensions utilisées au niveau industriel sont en général plus faibles. Le réseau du *WaferIC* est donc légèrement inférieur en taille au plus gros graphe utilisé pour les bancs d'essai des routeurs globaux pour *ASIC*. Il se situe en tous les cas parmi les plus importants. Ainsi, réaliser un routeur détaillé pour le *WaferIC* est similaire en terme de dimensions du graphe aux routeurs globaux pour *ASIC* publiés aujourd'hui. Le plus important réseau routier de la compétition *DIMACS* est clairement au dessus de tous. Les problèmes abordés dans cette compétition sont assez différents, étant donné la contrainte forte que représente un réseau irrégulier.

Dans cette introduction, le matériel sous-jacent au projet de recherche a été présenté. Il s'agit d'un projet qui impose des contraintes sur l'algorithme de routage connues (taille du problème, représentation de celui-ci, ...) et d'autres très nouvelles (architecture, temps de calcul, ...). Pour surmonter ces problèmes, cette thèse présente des contributions originales qui sont pour la majorité applicables à d'autres domaines. Les contributions originales de cette thèse sont :

- La méthode de calcul du plus court chemin dans un réseau multi-dimensionnel, de longueur d'arc exponentiels en $O(n)$, n étant le nombre de nœuds séparant la source et la destination à vol d'oiseaux . Il est prouvée mathématiquement que cette méthode est bien linéaire, et de moindre complexité qu'un algorithme de type labyrinthe.
- Une approche du routage dans un graphe régulier fondé sur une méthode aléatoire. Deux techniques en particulier sont proposées : le routage par permutations, et la résolution des congestions par une approche locale. Leur propriétés sont l'accélération par un facteur 3 à 100 des temps de calcul lorsqu'appliqués au *WaferIC* par rapport à un A^* utilisé seul, la parallélisation, la garantie d'optimalité des solutions générées pour une des deux techniques, la simplicité de mise en œuvre (réalisé par une seule personne en quelques semaines).
- Un algorithme d'équilibrage des délais, qui permet de garantir une différence de délai au sein d'un groupe arbitraire selon une contrainte spécifiée par l'utilisateur. Cet algorithme est une extension non-triviale de l'algorithme de gestion des délais courts appelé *Routing Cost Valley* (*RCV*) présent dans la littérature, et aurait de l'intérêt pour une application sur *FPGA*. Bien que des outils commerciaux proposent cette fonctionnalité, ceux-ci ne sont pas publiés ; le travail réalisé ici s'appuie sur la littérature existante pour l'étendre et supporter l'équilibrage entre des groupes de *net* de taille arbitraire.
- La découverte et démonstration que la loi de Rent, appliquée aux *PCB* avant 1980 avec succès et toujours d'actualité pour *FPGA* et *ASIC*, n'est pas utilisable comme métrique pour un modèle de *netlist* crédible pour les *PCB* modernes.
- Un modèle de *netlist* crédible pour *PCB*, qui permet de construire un générateur stochastique de *netlist*, une contribution originale là où les générateurs pour *FPGA* ou ASIC utilisent des techniques différentes basées

principalement sur de l'apprentissage.

– La réalisation d'un outil de routage fonctionnel au sein du *WaferConnect*, le logiciel complet qui permet l'utilisation du *WaferBoard*. Cet aspect représente une part importante du travail à caractère pratique.

Cette introduction a présenté le contexte des recherches et les principales contraintes qu'elles imposent à un algorithme de routage. Ces contraintes ont été comparées avec les technologies les plus proches et les plus pertinentes, ce qui permettra de mieux appréhender les choix réalisés pour l'algorithme de routage. De plus, il a été souligné que l'absence de bancs d'essais publics pour routeur *PCB* a conduit à la construction d'un modèle crédible de *netlist* pour *PCB*, et à la découverte de points majeurs que la communauté croyait acquise depuis de nombreuses années (loi de Rent).

Suite à la présentation du contexte des recherches, la thèse présente au chapitre 1 la problématique à laquelle elle s'attaque et la description des contributions réalisées. Au sein de la section 1.1, les contraintes détaillées du routage y sont énoncées. De plus, des comparaisons sont réalisées avec des technologies connexes pour mettre en perspective les contraintes et difficultés du problème. Ensuite, la section 1.2 énonce les contributions de niveau doctoral de la thèse et sont décrites succinctement. Des liens vers les chapitres de développement sont fait. À la section 1.3, une discussion de la portée des contributions est faite. Ceci permet au lecteur d'appréhender l'intérêt de chaque contribution et d'en évaluer leur pertinence.

Le chapitre 2 est dédié à une revue de littérature. Ce chapitre est scindé en deux grandes sections. En premier lieu, la section 2.1 effectue une revue en profondeur des outils de routages publiés dans la littérature. Nous y présentons les principaux domaines de recherches concernant le routage en général. Les diverses techniques et algorithmes qui représentent un intérêt pour l'application du WaferBoard sont ensuite détaillées. Deux grandes familles de routeurs en sont extraites. Tout d'abord, nous proposons une analyse des approches dites globales les plus importantes, et un historique des publications de ce type de routeur. Les approches dites séquentielles sont ensuite traitées de la même façon que les routeurs globaux. Les techniques de routage capables de garantir un équilibrage de délais au sein d'un groupe de *net* sont également présentées. Il s'agit d'une fonctionnalité spécifique de certains routeurs, et dont la littérature offre des ressources intéressantes. Enfin, une analyse des différentes approches de génération automatisée de *netlist* est faite. Leur intérêt et leurs applications y sont également mis en perspective, face à l'historique des publications de ce domaine.

Le chapitre 2 contient en deuxième lieu la section 2.2, où sont décrits les algorithmes majeurs sur lesquels les contributions développées au cours de ces recherches s'appuient. Les routeurs Dijkstra et A^* sont décrits en premier lieu. Puis, nous présentons la technique générale de routage séquentiel, dite « *Rip-up and reroute*. Suivront les deux algorithmes de routage pour *FPGA* majeurs présents dans la littérature, nommément *PathFinder* et *Versatile Place and Route* (*VPR*), que nous confronterons. Nous complèterons le chapitre 2 par la description de l'algorithme de gestion des délais courts publié le plus avancés sur *FPGA*, *Routing Cost Valley* (*RCV*), ainsi que la règle de Rent utilisée par de nombreux générateurs de *netlist*.

Suite à la revue de littérature, les chapitres suivants constituent le développement de la thèse. Au chapitre 2.2.6, l'article « *A Pattern-Based Routing Algorithm for a Novel Electronic System Prototyping Platform* » a été soumis le 15 juillet 2012 au journal *IEEE Transactions on CAD.* Il détaille les problématiques de routage sans équilibrage de délais. Nous y faisons la démonstration de la complexité algorithmique d'une technique de routage proposée : le calcul d'un plus court chemin (en terme de délai) entre deux nœuds arbitraires du *WaferIC*. De plus, nous proposons deux algorithmes d'accélération des calculs : le routage par permutation et le routage par sous-sections conflictuelles.

Le chapitre 4 présente les techniques développées pour l'équilibrage de délais. Nous aborderons les limitations de l'algorithme *RCV*, les besoins d'équilibrage pour le projet sont explicités. Nous proposons un algorithme fonctionnel, accompagné d'une technique d'accélération de routage. Ce chapitre constitue également le sujet d'un article en phase de finalisation d'écriture.

Enfin, le dernier chapitre de développement s'intéresse à la génération de *netlist* pour *PCB*, sous forme d'un article soumis à la revue *IEEE Transactions on CAD*. Il a été soumis dans une première version le 4 mars 2012. Une version corrigée a été soumise ensuite le 15 juillet 2012, suite aux commentaires constructifs des relecteurs. Les métriques utilisées pour les circuits *VLSI* et *FPGA* y sont décrites. Il est démontré qu'une des métriques n'est pas utilisable pour les circuits sur *PCB*, la règle de Rent. Enfin, un algorithme stochastique y est proposé, et les *netlist* synthétiques générées sont comparées à un ensemble réel.

Pour terminer, une conclusion termine la thèse et trace les perspectives des recherches futures concernant les techniques de routage élaborées au cours de ces recherches.

Chapitre 1

Problématique et Contributions

1.1 Problématique

Certaines des contraintes imposées à l'algorithme de routage proviennent de la structure matérielle du *WaferIC*. D'autres contraintes viennent du flot utilisateur du *WaferBoard*, et des outils qui gravitent autour. Pour utiliser le *WaferBoard*, l'utilisateur commence par placer les composants nécessaires à son circuit électronique sur le substrat programmable (cf. figure 0.0.1). Une fois le couvercle du *WaferBoard* assurant le contact entre substrat et composants fermé, c'est via un logiciel appelé *WaferConnect* que le *WaferIC* est configuré. Le système relié à l'ordinateur effectue d'abord une analyse des défectuosités présentes dans la tranche de silicium. Puis, il détecte et reconnaît les composants installés (sans restrictions de placement). L'utilisateur fournit un *netlist* source qui est dans un premier temps élaguée. Cette étape consiste à remplacer les composants passifs par leur équivalent, tels les résistances de tirage, les condensateurs de découplage, les connecteurs et les oscillateurs. L'algorithme de routage se charge ensuite du routage du *netlist* élagué en utilisant le réseau d'interconnexions du *WaferIC*. Le calcul de la configuration est réalisé par l'algorithme de routage qui doit être capable de rencontrer les contraintes spécifiées, en utilisant les ressources de routage disponibles. Enfin, les circuits posés sur le substrat sont mis sous tension. La figure 1.1.1 présente les grands aspects de l'utilisation du *WaferBoard*.

Le cycle de développement lors de l'utilisation du *WaferBoard* est très rapide, de l'ordre de quelques minutes entre le changement de placement des composants, incluant le démarrage du système, et la mise sous tension des composants. Cet aspect est capital pour que le prototypage avec le *WaferBoard* soit un succès commercial : les outils de la suite logicielle ont donc une forte contrainte de temps de calcul. En ce qui concerne l'algorithme de routage, l'objectif est de réaliser un routage en moins de 5 minutes pour des *netlist* de difficulté « moyenne ». Cette densité est établie à environ une bille tous les 4 nœuds (cellules). Il serait raisonnable de pouvoir router des *netlist* de forte densités (une bille toutes les 2 cellules) en moins de 5 minutes. Ces densités ont été établies suite à l'analyse de densité des *PCB* (cf. chapitre 4.4). Il s'agit d'une contrainte forte pour les recherches effectuées, qui se sont concentrées sur

plusieurs points au respect de cette contrainte. Sur *FPGA*, le placement et routage sont les opérations les plus coûteuses en terme de temps de calcul (pouvant aisément prendre plusieurs dizaines de minutes, voir quelques heures). Bien que tout concepteur *FPGA* accepterait volontiers des temps de calculs bien plus courts, celui-ci est à contraster avec le temps de développement d'un circuit sur *FPGA*, qui se compte au moins en semaines. Les quelques heures sont donc acceptables pour l'utilisateur, en particulier lorsque le *FPGA* est très fortement utilisé. De même, les développements de circuits *ASIC* prennent des semaines, voire des mois et les temps de calcul de placement routage peuvent prendre une dizaine d'heures, parfois plus [13]. Ce temps reste acceptable face aux mois de développement d'un *ASIC*. À contrario, le *WaferBoard* s'insère dans le flot de développement d'un système électronique ou *PCB*, et propose de diminuer drastiquement les temps de prototypages de ceux-ci. Les temps de calculs acceptables pour l'utilisateur se chiffrent donc en minutes.

Une particularité unique du *WaferBoard* est l'absence d'un placeur intelligent. Que ce soit sur *PCB*, sur *FPGA* ou *ASIC*, toutes ces technologies offrent un moyen de réduire les congestions par un placement très efficace. Les algorithmes de placement permettent de réduire fortement la proportion de longues connexions, ce qui soulage d'autant l'algorithme de routage. Le placement intelligent dans le cas du *WaferBoard* est fait par un utilisateur pressé de tester son nouveau système électronique, donc très loin de se soucier des aspects de localité. L'analyse du *netlist* d'entrée et du routage réalisé permettraient à terme de réaliser un algorithme de suggestion de placement à l'utilisateur. Une telle approche, bien que prometteuse, n'est pas réalisée aujourd'hui, et ce manque de placeur a des conséquences sur le réseau d'interconnexions. La forte connectivité du réseau soutient l'algorithme de routage pour compenser le manque d'algorithme de placement, mais surtout offre une tolérance aux pannes nécessaires pour tout circuit intégré à l'échelle de la tranche.

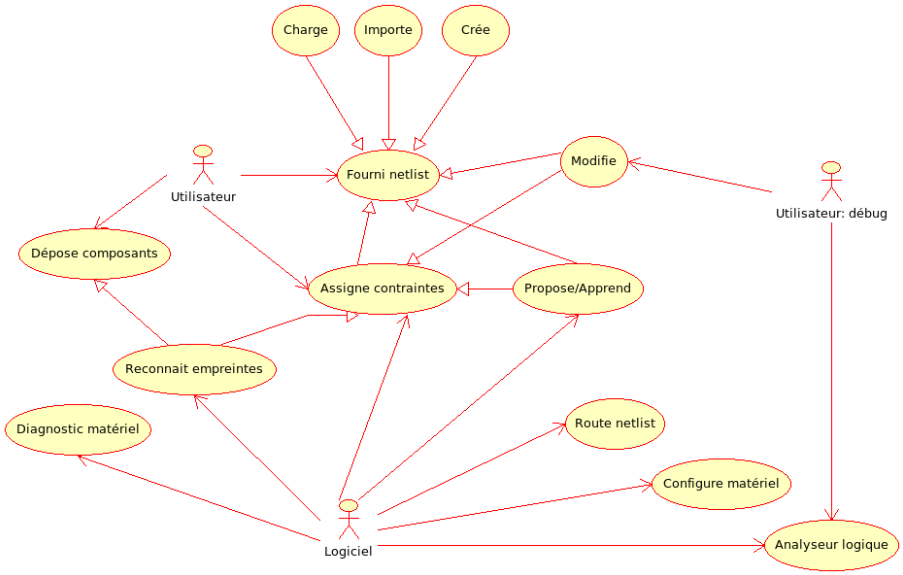


FIGURE 1.1.1: Cas d'utilisation à haut niveau du système de prototypage.

Pour bien comprendre certains choix de l'algorithme de routage, il est nécessaire de connaître le fonctionnement primaire d'un algorithme de routage, et l'impact du réseau sur celui-ci. Tout algorithme de routage réalise une modélisation du monde physique qui correspond à la problématique rencontrée. Pour un GPS, le calcul du plus court chemin se fait au travers d'une modélisation du réseau routier : un graphe qui connecte des nœuds (intersections entre routes) et des arcs ayant des poids variables (les routes elles-même avec différentes vitesses possibles - donc des poids différents). Pour un réseau routier, le graphe construit est non-régulier en général. En réalité, tout algorithme de routage réalise une représentation du monde au moyen d'un graphe. Pour un *PCB*, la taille et l'écartement minimal des pistes de métallisation défini la résolution de la grille qui sert de support au graphe. À chaque intersection de la grille, un nœud est présent (il est possible de changer de direction à chaque nœud). Les *PCB* étant multi-couches, le graphe construit est donc un réseau maillé régulier en 3 dimensions. Dans le domaine des ASICs, le graphe est très similaire. Par contre, certaines contraintes concernant le routage diffèrent : les vias sur *PCB* sont en général plus gros que la largeur d'une piste, la décision de placer un via doit se faire intelligemment. Sur ASIC, il existe pour certaines technologies des directions privilégiées [13]. Par exemple, la couche de métal 7 (niveau 7) obligerait toute piste à être horizontale, au contraire de la couche 6 qui serait verticale. Le graphe construit modéliserait donc cette réalité physique. Pour les *FPGA*, tout comme le routeur fruit des recherches pour le *WaferIC*, un graphe particulier modélise la réalité. En conséquences, il est très courant qu'un algorithme développé sur une technologie puisse être adapté sur une autre, moyennant quelques aménagements en fonction des différences de contraintes de routage.

En ce qui concerne les contraintes physiques, le réseau d'interconnexions est le premier sur la liste. Ce réseau, intégré au sein du circuit, est un réseau régulier multi-dimensionnel de degré 5, soit 24 arcs pour chaque nœud. Les arcs sont mono-directionnels, et donc doublés avec un dans chaque direction. Du fait de l'intégration à l'échelle de la tranche, la modélisation du réseau doit tenir compte de cet aspect physique. En effet, un grand nombre d'arcs ont été intégrés au *WaferIC* pour la raison majeure d'apporter une tolérance aux pannes. Ainsi, même une zone locale fortement touchée par des anomalies de fabrication permet de soutenir un composant dense posé au dessus. Au rang des raisons secondaires d'une telle densité, le placement des composants est loin d'être optimum car l'utilisateur souhaite passer peu de temps sur le WaferBoard, et cherche surtout à valider son application. Offrir une densité de connexions importante permet de compenser en partie pour ce manque. La taille du réseau est d'environ 9×10^5 nœuds, soit une taille comparable aux bancs d'essais très importants pour routeur globaux ASIC. Le nombre d'arcs associés est d'environ 4×10^6, soit un graphe plutôt dense par rapport aux autres technologies sur silicium. Ce réseau possède une particularité pour les routeurs en électronique : le délai minimum en fonction de la distance entre deux nœuds non-adjacents est une fonction non-monotone ; cette particularité est étudiée au chapitre 2.2.6. Ainsi, il n'est pas trivial de connaitre la route la plus courte pour une distance à parcourir donnée.

Le réseau d'interconnexions du *WaferIC* est environ 10 fois plus lent que les traces de cuivre sur *PCB* [14]. Ceci implique une latence nettement plus importante des signaux pour tout prototype utilisant le *WaferBoard*. Le routage

doit donc être orienté « plus court délai » pour limiter autant que faire se peut cet inconvénient. La figure 1.1.2 présente plusieurs graphiques du délai en fonction de la distance parcourue, pour un réseau multi-dimensionnel tel qu'intégré dans le *WaferIC*. Des simulations au niveau circuit, et une confirmation à l'aide d'un banc d'essai réalisé sur une puce de test, ont démontré que le délai d'un *crossbar* de 1733 ps et délai de franchissement d'une cellule de 230 ps. Les différences de délai dûes au placement et routage irrégulier du *crossbar* n'ont pû être mesurées, absorbées par les délais importants des multiplexeurs franchis. Cependant, il est possible d'étendre le modèle pour tenir compte des irrégularités du *crossbar*.

Chaque courbe de la figure 1.1.2 représente le délai du chemin le plus court pour différentes distances, en utilisant des réseaux de degré de plus en plus important. Par exemple, pour $D = 0$ l'arc le plus long est de $2^D = 2^0 = 1$ soit un réseau maillé classique. Dans ce cas et jusqu'à $D = 1$ la fonction est monotone. Par contre, dès que le délai de parcours d'un fil est supérieur à la moitié de celui du franchissement d'un *crossbar*, la fonction n'est plus monotone [1]. Prenons l'exemple d'un parcours d'une distance de 7 (cas simple), dans un réseau régulier offrant des liens de longueurs 1, 2, 4, et 8. Un chemin possible le plus court en terme de distance parcourue utilise les arcs de longueur (4, 2, 1) : son délai total au sein du *WaferIC* sera de $230 \times (4+2+1) + 1733 \times 3 = 6809$ ps. À contrario, le chemin le plus court en terme de délai utilise les arcs de longueur (8, 1) ce qui représente une distance parcourue totale de 9 (donc avec un dépassement). Le délai total est de $230 \times (8+1) + 1733 \times 2 = 5536$ ps, soit une réduction de plus de 40 %. Une des contributions de mes recherches a porté sur l'élaboration d'une solution mathématique qui permet de calculer le plus court chemin en terme de délai dans un graphe de degré D pour une distance arbitraire dans une complexité linéaire avec la distance à franchir, solution présentée au chapitre 2.2.6.

Les *netlist* utilisés sur le *WaferBoard* seront très similaires à celles sur *PCB*. Seuls les composants actifs sont admis, ce qui a pour effet de densifier les *netlist*. En effet, les condensateurs, résistances et les connecteurs occupent une surface importante, même sur des circuits récents, et les éliminer autorise un rapprochement entre composants de forte densité de connectivité (typiquement processeur, *FPGA*). À contrario, les *net* d'alimentation ne sont pas routés. Ceux-ci utilisent beaucoup de ressources de routage sur un *PCB*, car ils sont distribués auprès de tous les composants, sur de nombreuses broches. Au sein du *WaferIC*, les broches d'alimentation sont connectées aux grilles d'alimentation (au travers d'un régulateur) et non pas sur le réseau d'interconnexions. Cet aspect particulier mitige la densification des *netlist* sur le *WaferIC*.

En plus du routage « au plus court délai », l'algorithme de routage doit supporter une fonctionnalité dite d'équilibrage des délais. Dans un *netlist PCB*, des *net* peuvent aux besoins de l'ingénieur être regroupés ensemble pour former un bus. Au sein de celui-ci, il est courant de spécifier une contrainte d'équilibrage de délais, c'est à dire d'assurer que l'écart maximum du délai de propagation entre le plus court et le plus long au sein du groupe soin inférieur à une borne fixée (communément appelé biais de données ou *data skew*). Ces contraintes sont par exemple spécifiées pour des bus

1. La moitié car chaque longueur disponible est égale à la moitié de la longueur suivante.

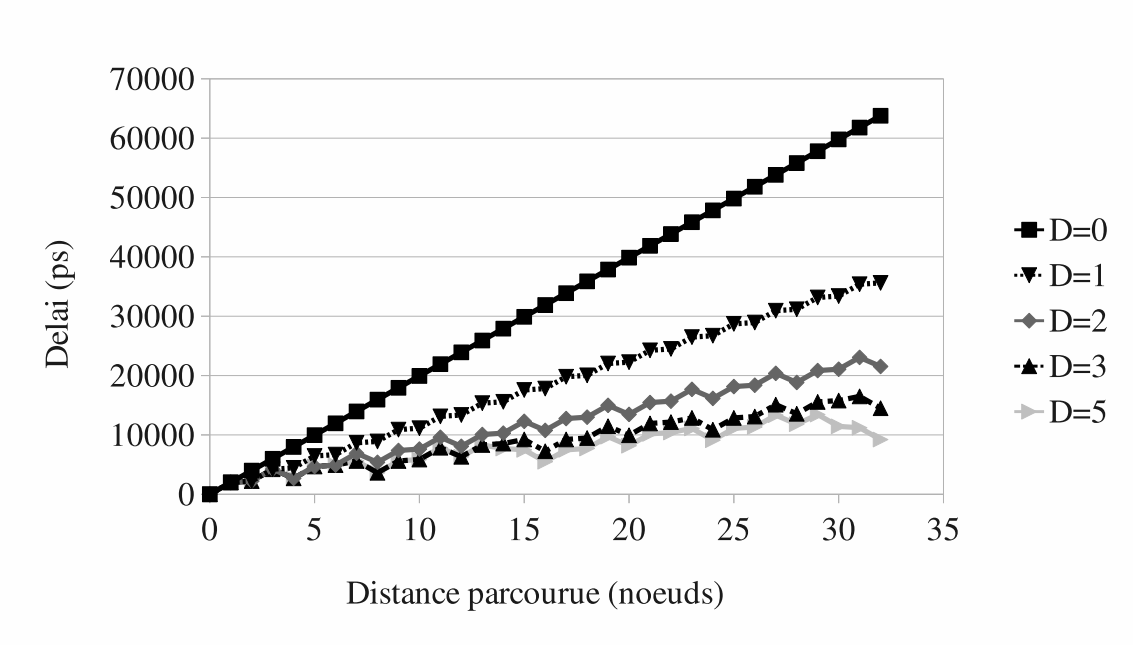


FIGURE 1.1.2: Délai en fonction de la distance parcourue, pour différents arcs choisi dans un réseau multi-dimensionnel régulier. D réfère à la dimension de l'arc. Par exemple $D = 0$ correspond à un arc de longueur $2^D = 2^0 = 1$ cellule (plus proche voisin). Dès que des arcs de longueur supérieure à 2 sont utilisés, la fonction du délai n'est plus monotone avec la distance parcourue.

de données entre un processeur et une mémoire ; les composants exigent que les fronts de données soient synchronisés avec une certaine tolérance (par exemple, 2 ns) pour rencontrer les spécifications temporelles de la communication. Dans ce cadre, un ingénieur spécifie cette contrainte sur le bus de données, et le routeur se doit de réaliser un routage qui assure une différence entre le plus court et le plus long délais inférieure à la contrainte. Étant donné qu'il s'agit d'une plateforme de prototypage, cette contrainte est prioritaire devant celle de la recherche du plus court délai que le routeur effectue pour chaque *net* individuellement. En effet, le circuit ne fonctionne tout simplement pas si la tolérance n'est pas respectée. Le routeur ne peut cependant pas réaliser d'équilibrage plus précis que ce que permet la technologie. Pour la version courante du *WaferIC*, les équilibrages sont réalisés par des détours des chemins, l'équilibrage est donc limité en pire cas par les délais des plus courts chemins réalisables (un *crossbar* plus le franchissement d'une cellule). Un équilibrage plus fin pourrait éventuellement être possible par l'insertion d'un circuit à délai programmable au niveau des entrées et sorties, mais ceci dépasse le cadre de cette thèse.

Une fonctionnalité très intéressante du *WaferBoard* est la possibilité à tout moment d'enregistrer une liste de signaux dans le temps, alors que le système électronique est fonctionnel. Pour soutenir cette fonctionnalité équivalent à un analyseur logique, l'algorithme de routage doit être capable de router des signaux à la demande au sein d'un *WaferIC* pré-configuré, avec un routage déjà réalisé. En effet, les signaux demandés par l'utilisateur doivent être copiés et acheminés vers un câble plat déposé sur la surface sensible du *WaferIC*, connecté à un *FPGA* disponible dans le *Top-PCB* ou simplement déposé sur le *WaferBoard*. L'algorithme doit donc à la demande réaliser le routage des signaux

demandés, au travers du *WaferIC* vers le *FPGA* mentionné, alors que les ressources de routage sont en partie utilisées. Cette fonctionnalité n'est pas un défi scientifique, mais elle pose quelques contraintes sur l'implémentation du routeur. L'écriture de l'ensemble des algorithmes de routage, réalisés à partir de zéro du fait des contraintes industrielles du projet, ne sont en réalité qu'une partie des efforts réalisés pour mener à bien ces recherches. En effet, j'ai également réalisé une majeure partie des structures utilisées pour l'outil global qui supporte le WaferBoard. Toutes les différentes briques qui composent le logiciel partagent les mêmes structures de données, avec des contraintes variées. L'analyse et l'architecture de toutes les briques, la description et l'écriture des structures de données partagées ont également dûes être implémentées au cours des recherches, en majorité par moi-même. Une grande partie du temps a donc été dédié à des sous-projets, certes non publiable, mais qui ont permis de construire un système cohérent et fonctionnel aujourd'hui.

La problématique de cette thèse est donc posée, et sa réalisation a mené à plusieurs publications et différentes approches originales. Certaines de ces approches sont applicables à d'autres technologies dès aujourd'hui, et dépassent largement le cadre du WaferBoard. La description des contributions scientifiques réalisées pour ces recherches est présentée à la section suivante.

1.2 Description des contributions

Les contributions décrites dans cette section ne sont pas ordonnées par importance, car il est délicat aujourd'hui de prédire lesquelles auront le plus d'impact à l'avenir. Elles sont ordonnées pour qu'une certaine logique s'en dégage, et ainsi en faciliter la compréhension.

1.2.1 Modèle et générateur de *netlist*

Pour caractériser tout algorithme, il est nécessaire de lui fournir des données d'entrées ; pour un algorithme de routage, il s'agit d'un *netlist* et d'un graphe. Le graphe est connu dans le cadre du *WaferIC*, par contre aucun *netlist* n'existe pour le WaferBoard, le produit n'existant pas encore. Les *netlist* nécessaires sont très similaires à celles sur *PCB*, cependant elles sont très difficiles à obtenir. De plus, des *netlist* configurables selon différentes métriques (densité, contraintes d'équilibrage, surface, ...) permettent de caractériser finement les algorithmes, mais n'existent pas pour *PCB*. Un chapitre de cette thèse est donc consacré à l'étude et la construction d'un modèle de *netlist*, dans l'objectif de créer un algorithme de génération de *netlist* de type *PCB*. Ces *netlist* synthétiques servent ensuite de données d'entrées à un algorithme de routage.

La technique de génération proposée est différente des approches proposées dans la littérature. Celles-ci sont subdivisées en deux grandes catégories. La première utilise un modèle des blocs à placer, qui sont générés aléatoirement puis interconnectés en utilisant principalement la loi de Rent. Cette technique est intéressante lorsqu'elle vise à produire des *netlist* non-placées (par exemple pour des outils de *floorplanning*) mais ne permet pas de gérer la distribution des

longueurs des *net* de façon précise. La deuxième approche utilise un *netlist* réel pour en construire une signature en fonction de différentes métriques. Puis, une série de mutants sont générés ; le générateur fait en sorte que la signature des mutants soit égale au *netlist* original. L'inconvénient majeur d'une telle solution est la nécessité d'obtenir un *netlist* réel pour chaque type généré, et de nombreux pour calibrer le modèle. De plus, les différentes métriques ne sont pas indépendantes les unes des autres, ce qui ne permet pas de caractériser les algorithmes rigoureusement. Par contre, l'avantage est de produire des *netlist* crédibles au sens des métriques utilisées.

Pour un générateur de *netlist* visant des routeurs *PCB*, il est très important de pouvoir maîtriser la distribution des longueurs des *net*. De plus, les *netlist PCB* placées sont très difficiles d'accès donc la technique de construction par mutation est peu praticable, surtout lorsque l'objectif est de construire un outil qui permettra de comparer les algorithmes de routage pour *PCB*. La solution proposée est une méthode stochastique qui calque les métriques visées. Les *netlist* sources et générés ont été routés par le même algorithme ; le taux d'utilisation des ressources est ensuite comparé. Les *netlist* réels obtenus de l'industrie sont variés en terme d'applications (pur transfert de données, gestion de données mélangé à de l'analogique, analogique presque pur) et en terme de surface et de technologies (de 2 à 16 couches). Les résultats montrent la pertinence de l'approche proposée car sur une vingtaine de *netlist* très variés conduisent à une utilisation de ressources très semblable du réseau de routage. Il est également montré que le modèle produit des *netlist* très crédibles pour un algorithme de routage, comparées à de véritables.

Une métrique très utilisée pour les générateurs de *netlist* pour *FPGA* et circuits *VLSI* s'appelle la règle de Rent. Une analyse réalisée et présentée dans cette thèse au chapitre 5.3 démontre que cette règle ne s'applique pas sur la technologie des *PCB* récents (après 2000), et ne peut être modélisée, puis servir de guide à un générateur. Ce résultat vient à l'encontre des premières études sur *PCB* faite dans les années 70. L'approche et l'algorithme ont été rassemblés dans un article soumis au journal *IEEE Transaction CAD*.

1.2.2 Techniques de routage point à point

Le calcul du plus court chemin est un problème résolu en un temps polynomial par les routeurs dit *maze* (en labyrinthe). Ceux-ci trouvent le plus court chemin dans un graphe irrégulier. Deux exemples très connus et souvent utilisés par les routeurs en *VLSI*/*FPGA* sont l'algorithme de Dijkstra et A*. Leur complexité dans le cadre d'un réseau relativement dense comme c'est le cas ici est en $O(|V^2|)$ avec $|V|$ le nombre de nœuds du graphe. Cependant, dans le réseau du *WaferIC* les temps de calculs sont importants. Il est possible de construire un algorithme capable de calculer le chemin le plus court (identifié comme le chemin qui offre le plus petit délai dans cette thèse) dans un graphe régulier en un temps qui croît linéairement avec le nombre de nœuds séparant source et destination ; ce n'est pas trivial comme montré au chapitre 2.2.6 pour un réseau dont les liens sont de type « puissance de deux » avec les contraintes connues du réseau utilisé, et constitue une des contributions de la thèse.

Dans cette thèse, la méthode de calcul du plus court chemin dans un réseau multi-dimensionnel régulier est accom-

pagnée d'une preuve mathématique de complexité, en $O(n)$ avec n le nombre de nœuds séparant la source et la destination et indépendamment du nombre de nœuds dans le réseau. Si la complexité algorithmique est nettement plus faible que celles des routeurs de type *maze*, c'est au prix d'une généricité moindre. La preuve présentée au chapitre 2.2.6 fait l'hypothèse d'un réseau dont les longueurs des arcs (fils reliant deux nœuds adjacents) sont des puissances de 2, bien qu'une piste de réflexion sur d'autres types de longueur soit mentionnée. Par contre, le degré du réseau, le rapport entre le délai de franchissement d'un nœud (*crossbar* dans le *WaferIC*) et celui du lien d'un nœud à l'autre sont laissés comme variable. Ainsi, il est démontré au chapitre 2.2.6 que le nombre maximum de routes calculées b_J dépend du degré du réseau uniquement, et croît comme la suite de Fibonacci. Le nombre total maximal de routes calculées par cette technique pour le plus court chemin entre deux nœuds est défini par les variables suivantes. Soit $\Gamma(v_s, v_d) = \frac{n}{m}\lambda = \frac{n}{m} \times 2^i \lambda_{min}$ la distance entre les deux nœuds, avec $(m,n) \in \mathbb{N}$ et λ_{min} l'arc le plus court du réseau. Soit J le nombre d'itérations réalisées par la méthode de calcul, il est démontré au chapitre 2.2.6 que :

$$\begin{cases} J = 1 & \frac{n}{m} \leq 0.75 \\ J = \left\lfloor \frac{\ln(m\lambda_{max}) - \ln(8n\lambda_{min})}{\ln(4)} \right\rfloor + 1 & m \geq 4,\ n \in \left]\frac{3}{4}m; m-1\right] \end{cases} \tag{1.2.1}$$

Le nombre de calculs dépend de la distance à parcourir (rapport de m et n) et des longueurs maximales et minimales disponibles dans le réseau. Le nombre total de routes calculées dans un réseau est démontré égal à b_J. Soit $\varphi = \frac{1+\sqrt{5}}{2}$, $\varphi' = -\frac{1}{\varphi}$, alors :

$$b_J = \left\lceil \frac{1}{\sqrt{5}} \left(\varphi^{J-1} - \varphi'^{J-1} \right) \right\rceil \tag{1.2.2}$$

Pour le réseau du *WaferIC* (longueurs minimale de 1 et maximale de 32), le nombre maximal de routes à calculer est de 3. Pour chaque route calculée, la méthode possède une complexité maximale de $O(\Gamma(v_s, v_d))$, étant donné que le nombre maximal d'itérations est fixe (et petit) pour une architecture de réseau donnée.

Une technique de routage aléatoire qui permet de grandement réduire les temps de routage face au A* est également proposée et constitue une contribution de cette thèse. Le routage aléatoire n'est pas utilisé dans la littérature, car la très large majorité des recherches se sont focalisées sur l'amélioration des techniques en situations de très fortes densité. Or, le routage aléatoire ne permet pas d'améliorer la qualité du routage dans des situations très denses ; cependant, il est montré dans cette thèse qu'il est possible de faire du routage aléatoire sans nuire à la qualité des solutions générées, et de réduire fortement les temps de calcul. La première technique proposée permet de résoudre rapidement un grand nombre de conflits, et ainsi de réduire le nombre de routes calculées par un routeur en labyrinthe (*maze router*), bien plus lent mais plus robuste. Pour tout algorithme de routage, une route est définie par une liste d'arcs. Cependant, il est également possible de la décrire sous la forme d'un point de départ (nœud), suivi d'une liste de couples longueur - direction, jusqu'à un point d'arrivé. En effet, chaque route au sein d'un réseau d'interconnexions de type maillé (mesh) multi-dimensionnel est constituée de « segments ». Chaque segment est en réalité un fil électrique particulier, qui ne

peut être partagé avec une autre route. Lorsque l'ordre des différents segments est changé, la route ainsi modifiée relie toujours les mêmes points de départs et d'arrivée mais n'utilise pas les fils électriques aux même endroits : il s'agit donc d'une technique de gestion des conflits. Elle est aléatoire car les permutations ne peuvent être guidées en fonction d'une heuristique, mais il est expérimentalement démontré dans cette thèse de l'efficacité d'une telle approche en terme de temps de calcul. De plus, le routage par permutation ne diminue pas la qualité du routage final, permet de guider la suite du routage pour gagner du temps sur les étapes ultérieures, et est simple à implémenter. Il est possible de permuter des couples très rapidement, chaque permutation étant une nouvelle solution potentielle pour une paire source - destination. L'algorithme impose la vérification de chaque route générée pour des conflits, ce qui peut être efficace avec l'aide de structures de données bien pensées. Au final, pour chaque route conflictuelle après premier routage, un nombre paramétrable de permutations sont testées pour chaque route (effort) ; dès qu'une solution est trouvée, celle-ci est enregistrée. La parallélisation du traitement des permutations a également été expérimentée, et donne un gain modéré jusqu'à 4 processus parallèles environ. Le comportement de cette approche aléatoire a été étudié sur de nombreux bancs d'essai. Cette approche et son analyse, couplée avec l'algorithme de routage en $O(n)$ et sa preuve formelle ont fait l'objet d'une soumission au journal *IEEE Transaction CAD*, et forme le chapitre 2.2.6.

1.2.3 Équilibrage des délais d'un groupe de *net*

Le support de l'équilibrage des délais a été largement étudié au sein des routeurs *VLSI* et *PCB* dans les années 1980 et plus. Cependant, l'équilibrage des délais possible dépend totalement du réseau d'interconnexions utilisé, car c'est sur celui-ci que les détours sont effectués. De ce point de vue, le réseau intégré dans les *FPGA* est bien plus près du *WaferIC* que celui dans les circuits *VLSI* et *PCB*. Cependant, la littérature est très restreinte dans ce domaine, bien que les outils commerciaux supportent cette fonctionnalité depuis de nombreuses années. Les recherches effectuées sur l'équilibrage des délais ont pris donc racines sur les quelques publications existantes pour *FPGA*, notamment l'algorithme *Routing Cost Valley* (*RCV*), qui permet de tenir compte de contraintes de délais maximum et minimum pour un *net* donné. Sur cet algorithme *RCV*, deux ajouts originaux sont proposés.

Le premier est la gestion des groupes. Lors du routage, chaque membre du groupe est routé en utilisant l'algorithme *RCV*, qui permet d'obtenir des routes dont le délai est compris dans une plage spécifiée par l'utilisateur. Cependant, il n'est pas toujours possible de router tous les membres du groupe dans le délai imparti, sans conflit de ressources. *RCV* est incapable d'équilibrer les délais dans ce cas. Or, dans le cadre d'une plateforme de prototypage telle que le *WaferBoard*, l'objectif premier est de tester et valider un circuit. En général lorsqu'un ingénieur précise des contraintes temporelles d'équilibrage, celles-ci sont nécessaires au bon fonctionnement du circuit (par exemple pour la communication entre une mémoire et un microprocesseur). Étant donné que l'équilibrage n'est pas forcément possible en une itération, il est nécessaire de construire un algorithme qui relaxe progressivement les délais maximum autorisés, pour assurer le fonctionnement du circuit. La latence augmente, mais l'équilibrage est respecté. La méthode de gestion de

la relaxation est une des contributions de cette thèse.

Conjointement à la gestion de la relaxation des délais dans les groupes de route, un algorithme aléatoire de recherche de solutions équilibrées a été exploré pour réduire significativement les temps de calcul. En effet, sans cette accélération les temps de calculs se chiffrent parfois à plusieurs dizaines de minutes, trop élevé pour le WaferBoard. L'approche proposée consiste à pré-construire une table de solutions pour l'allongement du délai, étant donné qu'en première passe chaque *net* est routé selon un délai optimal. Pour allonger le délai d'une route au sein d'un réseau d'interconnexions tel que le *WaferIC*, il n'est pas forcément efficace d'allonger la distance franchie. Il est cependant possible d'allonger le délai d'une route en découpant un lien d'une longueur λ par deux liens de longueur $\frac{\lambda}{2}$. Il s'agit d'une propriété des réseaux d'interconnexions *CMOS*, qui sont l'allongement du délai d'une route par l'utilisation de plusieurs segments plus courts, au lieu d'un seul. D'autres patrons de construction sont également possibles, et décrits au chapitre 4. Cette approche aléatoire utilise plus de ressources de routage (c'est à dire plus de fils) qu'un A^*, mais sa complexité est linéaire au lieu de quadratique, et des implémentations très efficaces sont possibles. Au travers de nombreuses expérimentations, cette technique est montrée efficace lorsqu'appliquée au *WaferBoard*, et pourrait éventuellement être adapté aux routeurs pour les *FPGA*, circuits *VLSI* et *PCB*.

Enfin, il est possible d'accélérer encore le routage en utilisant la technique des permutations pour chaque allongement testé, multipliant les chances de trouver rapidement une solution valide, c'est à dire non-conflictuelle et dans les contraintes de temps spécifiées.

1.3 Discussion

Les contributions présentées ci-avant découlent du contexte des recherches, des contraintes du projet, des idées qui sortent de la littérature aujourd'hui. Une large part des contributions sont applicables à divers algorithmes de routage dans d'autres domaines, cependant leur intérêt mérite d'être discuté.

Avant toutes choses, la première contrainte importante des recherches effectuées proviennent du réseau d'interconnexions. Celui-ci partage des similitudes fortes avec ceux embarqués sur *FPGA*, mais conservent des spécificités qui rend difficile, voire impossible, toute comparaison directe des résultats de routage obtenus. Ce réseau a été bâti pour répondre à certains critères, notamment la surface occupée, la densité des interconnexions, la longueur maximum des liens, la tolérance aux pannes, l'intégration à l'échelle de la tranche. Au cœur du réseau se situe le *crossbar*, qui a été étudié principalement par l'auteur de cette thèse. Les résultats de cette étude ont été publiés à la conférence *NEWCAS* 2008 [15]. Ce réseau a été ensuite testé sur une puce fabriquée spécialement pour une version miniature du *WaferIC* complet. Cette version nommée *TestChip* V1.0 comportait une matrice de 3×3 cellules, et les liens entre cellules de courtes distances (1 et 2) ont pu être testés et validés. C'est par ce moyen, ainsi que par simulation après placement que les délais d'interconnexions ont été extraits. Ces résultats ont été publiés à la conférence *MNRC* 2009 [14] sous forme d'article, et a obtenu le deuxième prix au « *Best Paper Award* » de la conférence.

Du côté des contraintes de temps de calcul, d'occupation mémoire et d'implémentation, une analyse détaillée du flot de travail du *WaferBoard* a été réalisée avant les recherches spécifiques au routage. Ainsi, il a été démontré de l'importance de temps de calcul restreints ; également, les interactions du routeur avec les autres outils logiciels pour le WaferBoard ont été déterminés. Ces travaux ont été publiés à la conférence *ISCAS* 2008 [16]. Ces travaux préliminaires ont permis de bien comprendre le contexte du projet, les contraintes uniques qui s'y appliquent ainsi que les points communs avec d'autres technologies. Les choix algorithmiques et architecturaux prennent racines dans les recherches qui ont mené à la publication de ces trois articles de conférences.

En ce qui concerne directement l'algorithme de routage, une des contributions majeure est le routage par l'aléatoire. Il sera montré dans la revue de littérature du chapitre 2 que ce type d'approche n'a pas été étudié. Or, il est démontré dans cette thèse que l'accélération du routage peut atteindre plus d'un ordre de grandeur : ce résultat majeur étonne, et remet sans doute en question sa crédibilité. Par exemple, serait-il possible d'utiliser une telle approche et ainsi diviser par 10 les temps de routage sur *FPGA* ? Il s'avère que reproduire les résultats obtenus sur un circuit intégré à l'échelle de la tranche, auprès d'un circuit de quelques centaines de millimètres carrés n'est pas aussi simple. Premièrement, les techniques de routage aléatoire proposées ne sont pas totalement aléatoires : elles sont guidées par construction, c'est à dire que l'espace de recherche est défini. Cet espace donne aux solutions potentielles générées une garantie d'optimalité, et un taux de succès élevé, ce qui donne d'autant plus d'efficacité à une telle approche. Il est tout à fait possible d'utiliser une telle méthodologie auprès d'autres technologies.

Cependant, un certain nombre de spécificités du *WaferIC* n'autoriseront probablement pas de gains aussi importants sur *PCB*, *VLSI*, *FPGA*. Toutes ces technologies utilisent un algorithme de placement intelligent (bien que sur *PCB* il s'agisse souvent d'un ingénieur). Ce placement permet de très largement réduire les pics de la demande en ressources de routage. Un étalement intelligent des composants permet donc de réduire les ressources de routages nécessaires. Or, ce placement n'existe pas pour le *WaferBoard* : pour compenser, une forte densité d'interconnexions est mis à la disposition du routage, pour faire face à un maximum de conditions de placement. À cela s'ajoute les problèmes de tolérance aux pannes. Du fait de l'intégration à l'échelle de la tranche, le réseau d'interconnexions doit autoriser des routages importants, quand bien même une partie des ressources sur une zone soit non-fonctionnelle après fabrication. Le réseau actuel ne permet bien entendu pas de circonvenir à toutes les situations imaginables, cependant l'objectif était lors des phases de conception du matériel d'utiliser tout l'espace disponible en ressources supplémentaires. Les taux de défectuosités ne sont pas connus précisément, et les utilisations du WaferBoard étant très variés, le réseau est fortement connecté. Les approches aléatoires sont rapides lorsque la probabilité de générer une solution viable est importante. Or, si localement la densité des demande en ressources est par endroit importante sur le *WaferIC*, globalement elle est plus faible que sur *FPGA* ou *VLSI*. Cet aspect aide énormément au succès des approches aléatoires appliquées au *WaferIC*. L'effet de la distribution et du taux de défectuosités sur les performances du routeur n'a pas encore été étudié.

Comparativement aux bancs d'essais pour *VLSI* ou *FPGA*, le routage pour *WaferIC* semble donc facile, car le rap-

port entre la demande en ressources et la disponibilité des ressources est faible. Cependant, deux contraintes spécifiques contre-balancent ceci. La première est la forte connectivité du réseau, cumulée aux distances importantes parcourues par les *net*. Les routeurs en labyrinthe performants, tels le A*, ont une complexité fonction du nombre de nœuds et d'arcs. Le nombre important d'arcs au sein du *WaferIC* ralentit le A* dans les zones congestionnées. En effet, sur le réseau du *WaferIC*, le nombre de routes possibles est exponentiel avec la distance à parcourir. Deuxièmement, le temps de calculs acceptable pour le *WaferIC* est très petit, de l'ordre de la minute. Sur *FPGA*, *VLSI* ou *PCB*, les recherches se sont au contraire focalisées sur le routage dans des conditions difficiles (congestions particulièrement importantes) et sur l'amélioration de la qualité des résultats, plutôt que sur la diminution des temps de calcul, moins prioritaires.

De manière générale, les algorithmes proposés dans cette thèse sont applicables à d'autres technologies. L'impact des techniques de routage aléatoire est très certainement plus faible que sur le *WaferIC*, mais méritent d'être étudiées, notamment sur *FPGA*. Par contre, les recherches concernant le modèle de *netlist* pour *PCB* embrassent une problématique plus large que le *WaferBoard* lui-même, et constituent une brique intéressante pour la recherche. Cette section de la thèse (chapitre 5) a mené également à la découverte de limitations de la loi de Rent sur *PCB*.

Chapitre 2

Revue de littérature

Ce chapitre a pour objectif de présenter une revue de littérature des différentes techniques de routage dans le domaine de la micro-électronique. Cette revue permettra de justifier les choix qui ont conduit vers les algorithmes destinés au WaferBoard. Ainsi, les techniques de routage par recuit simulé, d'équilibrage des délais et les techniques de générations de *netlist* sont présentées. Dans un second volet, les algorithmes de routage significatifs pour cette thèse sont décrits plus en détail. Les algorithmes qui sont directement utilisés, et parfois étendus, sont également décrits en détails, ce qui permettra de comprendre les contributions de cette thèse.

2.1 Publications

2.1.1 Routage : les différentes approches explorées

L'objectif de cette revue de littérature consiste à dresser une cartographie des principales catégories d'algorithmes, associées aux problèmes majeurs approchés par les outils *CAD* de routage. Cette revue se veut large pour mieux cerner la problématique identifiée dans le cadre du projet de recherche *DreamWafer*, de ses spécificités et des techniques les plus prometteuses pour un algorithme de routage pour le *WaferBoard*™.

2.1.1.1 Les principaux domaines de recherche

Cette sous section donne une vue très générale des grands domaines de recherches en ce qui concerne les techniques et algorithmes de routage. Pour en faciliter la lecture, les références ne sont pas données : toutes les affirmations suivantes sont étayées aux sections 2.1.1.2, 2.1.1.3 et 2.1.1.4 et contiennent les références.

La problématique du routage est présente dans de nombreux domaines. Chacun possède des spécificités propres, cependant il y a eu de nombreux recoupements au fil des années. Ainsi, des mathématiciens se sont penchés sur des problèmes de routage et ont construit notamment la théorie des graphes ; la problématique de routage est apparue très tôt dans le domaine des *PCB* (« *Printed Circuit Board* »). Aujourd'hui les routeurs pour *PCB* évoluent peu, bien que

certaines contraintes d'équilibrage aient été récemment abordées . Le domaine du *VLSI* (*Very Large Scale Integrated Circuit*) et des routeurs globaux et de détail ont réalisés d'énormes progrès tout au long des 35 dernières années. Les routeurs globaux constituent une famille de routeurs (en particulier dans le *VLSI*) qui réalisent les connexions entre grandes zones du graphe sans se soucier de chaque *net* à l'intérieur de chaque zone. Les routeurs de détail réalisent les connections entre chaque *net*, dans un espace attribué. La recherche scientifique n'a cependant pas été continue, avec un ralentissement net entre 1995 et 2005 : la reprise des compétitions (*International Symposium On Physical Design - ISPD*) pour les routeurs de ce type en 2007 et 2008 a apporté un regain d'intérêt, notamment dû à la très forte augmentation de la problématique de la variabilité des délais d'interconnexions présents dans les procédés de fabrications nanométriques.

Les routeurs pour *FPGA* ont explosés eux durant le milieu des années 1990, avec les premiers algorithmes de routage orientés gestion des congestions (« congestion driven »). Aujourd'hui moins développés, c'est le domaine du routage de paquets (qualité de service notamment) dans les réseaux de télécommunications, dont les contraintes sont parfois assez différentes, qui attire beaucoup de recherches. La taille du réseau est certes nettement plus faible, mais les temps de calculs très limités et les besoins colossaux. Il s'agit aujourd'hui, avec les problèmes de routage pour les réseaux sans-fil, du domaine le plus prolifique en terme de publications.

2.1.1.2 Classification des algorithmes de routage

Les principales familles de routeurs ayant un potentiel intérêt pour cette thèse sont d'abord présentés ; ensuite une synthèse des points forts et faibles pour l'application visée est faite, justifiant les choix réalisés pour les expérimentations et implémentations.

Le problème de routage n'est pas nouveau : il a été étudié dans de très nombreux problèmes, allant du fameux « voyageur de commerce [17] » au *k*-plus courts chemins [18], en passant par la gestion de trafic dynamique (réseaux de télécommunications, qualité de service, réseaux routiers, ...). Le problème du voyageur de commerce est l'un des plus anciens connu dans le domaine du routage, sa première formulation mathématique connue remonte à 1930 [17]. La plupart des problèmes de routage sont NP-difficiles et ont conduit depuis à de très nombreuses recherches, tant sur des aspects purement théoriques, que sur des heuristiques spécialisées.

De manière générale, deux types d'algorithmes se distinguent : ceux cherchant une réponse optimale (ou au moins bornée) et ceux cherchant une « solution acceptable » sans garantie de qualité.

Les deux approches s'appuient sur la théorie des graphes. Pour les problèmes les moins difficiles (algorithme de complexité polynomiale) dont les plus courants sont rapportés au Tableau 2.1, des algorithmes séquentiels (qui résolvent le problème un *net* à la fois) peuvent être utilisés de façon efficace. Les plus connus sont Dijkstra [19] et A* [20], mais il en existe d'autres (Prim [21], Kruskal [22], ...). Pour les problèmes dits NP-difficiles, énumérer toutes les possibilités n'est pas efficace dans la plupart des cas, bien que quelques exceptions existent, précisées dans les sections

suivantes. Ainsi, des algorithmes capables de trouver une solution optimale à plusieurs problèmes d'optimisation existent dans le domaine du routage, et utilisent souvent des approches globales. Ils formulent le problème et tentent de résoudre le système construit, souvent par itération. Les plus connus sont k-SAT [23], la programmation linéaire [24] (proche de la programmation entière [25]), certaines approches utilisant le multiplicateur de Lagrange [26]. Ces algorithmes peuvent être efficaces et offrent souvent des possibilités de parallélisation importantes [27, 4, 28, 29]. Dans cette catégorie, sont également inclus les approches fournissant une garantie de qualité (borne supérieure et inférieure de qualité de routage). Le multiplicateur de Lagrange est un exemple typique d'algorithme capable de fournir une telle garantie, et donnant une grande souplesse entre temps de calcul et qualité des solutions générées [1], surtout lorsque la fonction de coût est convexe. La fonction de coût guide l'algorithme dans la recherche d'une solution, et permet de mesurer la qualité de chaque tentative. Les algorithmes de ce genre ont cependant de gros problèmes de temps de calcul, la garantie de qualité demande un gros effort de modélisation, et malheureusement beaucoup de calculs.

De l'autre côté du spectre des approches, les heuristiques visent à fournir une solution « satisfaisante » pour un problème, un contexte souvent plus spécifique. Les heuristiques sont très nombreuses dans le cadre séquentiel (une route à la fois, triées selon une heuristique établie). La technique du recuit simulé [30, 31] y est très présente, souvent associé à un routeur de type labyrinthe (*maze*) [32]. Le recuit simulé autorise la recherche de solutions à s'éloigner du minimum local courant, dans l'espoir de trouver dans l'espace des solutions un minimum plus bas que celui actuellement trouvé. À chaque itération, l'algorithme autorise des solutions de moins en moins éloignées du minimum local (on parle de réduction de « température » qui restreint l'espace de recherche). Une des problématiques majeures rencontrées par les chercheurs étant la gestion des congestions, des heuristiques de prédiction et d'estimation de la congestion ont été proposées [33, 34, 35, 36, 37, 38]. Cependant, ces heuristiques sont très délicates et souvent complexes à implémenter : il est difficile de compenser le temps de calcul de l'heuristique par un tri des *net* plus efficace. Les heuristiques séquentielles offrent surtout un temps de calcul clairement plus court que les approches globales [39]. Il existe également des heuristiques pour des approches globales, mais elles sont moins courantes. Elles sont notamment utilisées dans le domaine de la qualité de service (routage adaptatif, par exemple *TCP/IP*), par leur taille de problème relativement limité en comparaison au *VLSI*, au besoin de qualité, mais en réponse à des ressources de calculs et temps de calculs autorisés limités [8, 1]. Les formulations utilisées dans ce domaine ne sont pas applicables au routage statique demandé pour l'application ; de toutes manières l'échelle du réseau d'interconnexions du *WaferIC* est trop importante pour que ces techniques soient applicables avec des temps de calcul raisonnable.

De manière générale, chaque approche est plus ou moins adaptée à un problème particulier. Les outils pour le *VLSI* et *PCB*, ou le moindre pourcent de gagné en fréquence, dissipation de puissance, longueur de fil, ... est souvent acceptable au prix de longues heures de calculs : dans ces domaines, les algorithmes sont orientés qualité et de nombreuses approches sont globales. Bien que les *FPGA* soient conçus pour le prototypage rapide, il est possible aujourd'hui d'obtenir des *FPGA* performants à prix compétitif pour être utilisés dans des produits vendus à grande échelle. Les

Tableau 2.1: Les principales familles de routeur les plus couramments utilisés par les outils CAD en microélectronique.

NP-difficile	Polynomial
k-plus courts chemins	Plus court chemin
Plus court chemin avec au moins 2 contraintes	Plus court chemin avec 1 contrainte
Arbre Steiner (en général)	Arbre couvrant minimal

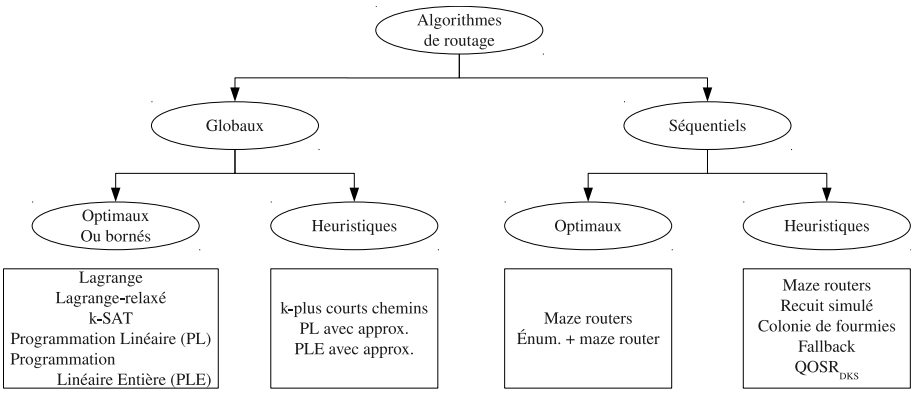


FIGURE 2.1.1: Essai de classification des différentes techniques de routage existantes. Quelques exemples d'algorithmes pour chaque catégorie sont donnés. Quelques références, qui seront détaillées plus loin : Lagrange [1], Lagrange relaxé [2, 3], Programmation Linéaire [4] (avec approximations [5]), Programmation Linéaire Entière [6] (avec approximations [7]), $QOSR_{DKS}$ [8].

recherches académiques les plus récentes sur les algorithmes de routage pour *FPGA* ont également mis l'accent sur la gestion de la congestion, tout en utilisant des heuristiques séquentielles [40, 41, 42]. Pour la plateforme de prototypage rapide traités dans cette thèse, un cycle de développement rapide est une contrainte importante. Un prototype n'ayant pas besoin d'être aussi performant qu'un produit final (quelle que soit la métrique utilisée), les algorithmes utilisés sont basés en général sur des heuristiques séquentielles. La figure 2.1.1 présente les grandes approches citées, dans une classification qui est généralement admise dans la communauté.

2.1.1.3 Approches dites globales

Les approches dites globales sont utilisées pour la qualité des solutions générées, ou la garantie de qualité. À l'origine utilisés dans les domaines acceptant des temps de calculs importants, des percées récentes ont démontré que les temps de calcul peuvent être compétitifs avec les approches séquentielles dans certaines conditions.

Une méthode très utilisée par les algorithmes de routage est LARAC [3][43] et ses versions dérivées [1][2]. Ces algorithmes utilisent le multiplicateur de Lagrange pour optimiser chaque route en fonction d'une contrainte de délai et de coût. Le multiplicateur de Lagrange, noté λ, permet de rechercher un optimum ou une solution de qualité bornée

selon une fonction objectif du problème. Plus précisemment, l'objectif est de minimiser m_λ tel que :

$$m_\lambda = c + \lambda d \tag{2.1.1}$$

ou c représente le coût d'un chemin et d sont délai. Le multiplicateur de Lagrange permet de rechercher m_λ minimal, et les différentes heuristiques adoptent une technique de calcul de coût ainsi qu'une approche spécifique sur la modulation du paramètre λ à chaque itération. Cette approche, très élégante et générique, souffre d'une complexité pratique importante mais permet de générer de très bonnes solutions. De plus, il a été démontré que les problèmes nécessitant une fonction de coût complexe (plusieurs métriques) nécessite également soit une optimisation pour chaque métrique [44], soit une combinaison linéaire [45]. Cependant, la méthode du multiplicateur de Lagrange devient dans ce cas très sensible aux poids assignés [44], et n'est efficace que dans le cas d'une fonction convexe, ce qui est rarement le cas dans le cadre des problèmes de routage [46]. Cette technique offre de bons résultats et les temps de calculs restent acceptables pour des applications de routage dynamique (*QoS* par exemple [8]) dans les réseaux de télécommunications, et plus récemment pour les routeurs globaux en *VLSI*. Leur utilisation reste marginale, d'une part pour des raisons historiques, et d'autre parts en raison de la complexité de l'implémentation et de sa robustesse.

Une approche similaire au multiplicateur de Lagrange utilise les k-plus courts chemins [46, 47] (k-SP) pour améliorer l'heuristique d'optimisation. En effet, la méthode de Lagrange classique retient en général un ensemble de chemins de la source vers la destination, pour sélectionner le meilleur (minimisation de l'équation 2.1.1). L'approche des k-SP enregistre les k plus courts chemins de u vers d, avec u le nœud courant (intermédiaire entre le nœud source s et le nœud destination d). Un tri de cette liste est effectué pour chaque nœud et itération, ce qui augmente la complexité et augmente la qualité des solutions générées. En général, ces algorithmes utilisent une condition de sortie relaxée par rapport à l'approche classique, dont le surcoût en terme de temps de calcul est faible [46], voire les réduis pour certains problèmes [48]. Ces algorithmes offrent une complexité similaire en pratique au multiplicateur de Lagrange, et permet d'améliorer les solutions générées. Ils souffrent d'une consommation mémoire nettement supérieure, ce qui peut poser des problèmes, par exemple pour des applications *VLSI*. Ils sont utilisés en général dans des graphes de quelques centaines de nœuds, plus difficilement sur des milliers de nœuds. Le *WaferIC* est clairement dans la catégorie *VLSI* en terme d'échelle, ces techniques seraient donc très difficiles à appliquer pour le WaferBoard.

La programmation linéaire (PL) est une méthode mathématique pour optimiser (maximiser ou minimiser) une fonction objectif (aussi appelée coût) linéaire, servant de modèle au problème rencontré. Cependant, la formulation mathématique est sensiblement différente du multiplicateur de Lagrange [24]. S'il est possible de représenter un problème donné par une fonction de coût linéaire dont la minimisation (ou maximisation) implique une solution optimale au problème, alors la PL peut être appliquée. Cette méthode est utilisée pour l'optimisation de routage sous contraintes, notamment pour la qualité de service (*QoS*) [49]. En effet, on peut modéliser la contrainte de délai (*QoS*) dans le cadre d'un routage de k-flots (un flot représente une suite de paquets qui ont une même origine et une même destination) dans un réseau d'une capacité c selon la formulation mathématique de programmation linéaire. Récemment, un algorithme

de complexité polynomiale capable de proposer des solutions de qualité garantie selon une borne donnée, a été proposé [5, 50]. En pratique, cette approche permet d'égaler les temps de calculs de [51], qui était le premier algorithme de programmation linéaire indépendant du nombre de flots à router. L'application de cette technique pour des problèmes de routage statiques (*VLSI*, *FPGA*, ...) passe par la programmation linéaire entière (PLE), et qui pourrait en théorie constituer une approche pour l'application visée.

La programmation linéaire entière (PLE) est un dérivé de la PL. Le problème est représenté de la même manière, mais les variables de contraintes sont uniquement des entiers : cette seule différence suffit pour rendre le problème NP-difficile dans la majorité des cas [6]. Le problème de routage des k-plus courts chemins dans un graphe est typique d'une formulation PLE. En effet, le choix de chaque arc pour chaque arbre (nœud électrique) est représenté par un entier (en fait un booléean ; 0 non utilisé, 1 utilisé). Les différentes contraintes sont représentées dans le problème par le même moyen. On peut dénombrer au moins deux approches différentes, la première résoud le problème PLE directement [52] (un exemple récent [42]), la deuxième relaxe le problème sous la forme de Programmation Linéaire [25] (un exemple récent [53]). Ainsi, la fonction objectif est autorisée à prendre des valeurs non-entières et le choix final est réalisé via un arrondi de la meilleure solution. Ces techniques ont récemment vu un regain d'intérêt important dans la communauté, pour des algorithmes de routage globaux en *VLSI* (2006-2009). En effet, une compétition nommée ISPD [54, 13] a permis à des équipes du monde entier à confronter leurs approches sur un ensemble de bancs d'essais, sur deux années consécutives (2007-2008). Cette compétition a permis de grandes avancées dans la qualité et la robustesse des algorithmes de routage globaux et la publication de nombreuses approches, dont les meilleurs résultats ont été obtenus avec des algorithmes basés sur la programmation linéaire entière [55] en 2007, mais battus par la suite en 2008 par des routeurs séquentiels qui offrent également des temps de routage bien plus courts. Or, le temps de routage est un point critique pour le routeur du projet.

Les approches globales ont été étudiées dès la naissance des outils *CAD* et de la micro-informatique. Bien que leurs performances aient été largement éclipsées par les approches séquentielles, un très net regain d'intérêt pour ces techniques est apparu pendant les années 2000. Avec l'augmentation de la puissance de calcul, et surtout l'importance des interconnexions avec les technologies à l'échelle nanométrique, le besoin de routeurs capables de fournir des solutions de très grandes qualité est devenu important. Les plateformes de prototypages (*FPGA* et le *WaferBoard*) restent l'apanage des solutions séquentielles, mais de très bons algorithmes de cette famille existent pour les routeur globaux en *VLSI*, comme étudiés dans la section suivante. Il a été montré au chapitre 1 que les dimensions du réseau d'interconnexions du *WaferIC* sont comparables aux plus grand bancs d'essais publiés pour le routage *VLSI*. De plus, les fortes contraintes de temps de calcul laissent peu de place aux algorithmes les plus lents, et ce quand bien même ils génèrent des solutions de meilleure qualité. De plus, le côté prototypage du WaferBoard n'exige pas le meilleur en terme de qualité des solutions, et c'est pour toutes ces raisons que les approches dites globales sont moins intéressantes que les routeurs séquentiels, dont la complexité algorithmique est nettement moins sensible à la taille du réseau.

2.1.1.4 Approches dites séquentielles

Les algorithmes séquentiels se basent essentiellement sur une approche « Enlève et re-route » (« *rip-up and re-route* ») ou R&R [56, 57]. Les heuristiques étudiées cherchent à définir principalement l'ordre dans lequel les routes sont calculées, les techniques de calcul de congestion, et l'estimation de la « marge de manœuvre » (« slack ») [58, 59, 60, 34, 38]. La première force de cette famille d'algorithmes est sa complexité de calcul pratique : ils sont clairement plus rapides que l'approche d'optimisation globale. Ils sont donc très utilisés pour les plateformes de prototypage (principalement les outils pour *FPGA*). Dans le domaine du *VLSI*, ils sont les plus utilisés, et sont les vainqueurs des compétitions *ISPD* [61, 41].

Le « recuit simulé » (« *simulated annealing* ») est une approche qui a fait ses preuves dans les algorithmes de routage pour *FPGA*, à partir du milieu des années 1990. Les algorithmes de routage efficaces pour la gestion de la congestion des ressources de routage, tels *Versatile Place and Route* (*VPR* [62, 63]), *PathFinder* [64] et leurs évolutions [65, 66] attaquent les problèmes de congestion en associant un coût à chaque ressource utilisée, après un premier routage sans gestion de conflits. Ce coût permet, via la technique de « recuit simulé », d'assigner les ressources les plus demandées (i.e. au coût le plus élevé) aux chemins les plus critiques.

De manière générale, les algorithmes de gestion de congestion décomposent le routage en quelques étapes [64] :

1. Router chaque chemin sans tenir compte des conflits ;
2. Associer à chaque ressource un coût proportionnel à leur demande (ie. nombre de conflits) ;
3. Pour chaque chemin en conflit, router ce dernier à nouveau à l'aide d'un parcours d'arbre, c'est-à-dire le parcours des ressources de routage. Chaque nœud est choisi en fonction d'une ou plusieurs métriques, fonction notamment du délai restant (comparé aux contraintes), des congestions environnantes, et souvent d'autres métriques ;
4. Itérer sur 3 selon une certaine condition d'arrêt telles que contraintes non résolues, « température » trop basse.

Ces algorithmes ont donc besoin d'une heuristique pour calculer la criticité des chemins. Celle-ci est en général réalisée par la combinaison d'une analyse temporelle statique [67, 68], de la liste des contraintes fournies par l'utilisateur (par exemple objectifs de fréquence d'opération du circuit, de puissance consommée) mais aussi par une heuristique d'allocation de la « marge de manœuvre » existant entre le délai estimé à priori et les contraintes de délai maximal (*slack allocation*). Cette heuristique est appliquée en général entre les étapes précédentes 2 et 3, et diffère selon les algorithmes. Elle permet à ceux-ci de ne pas calculer l'ensemble des chemins possible (très coûteux en calcul) mais de faire des choix entre les différentes branches lors de la gestion des congestions. Cette approche est notamment utilisée par les outils commerciaux des grandes entreprises (Xilinx, Altera) mais également au sein d'outils *CAD VLSI* (Cadence, Mentor Graphics). On peut également noter le travail de Ozdal et al. [69, 70], qui ont travaillé notamment sur des algorithmes de routages avec équilibrage des délais (cf. section 2.1.3), mais ont proposé une approche avec gestion efficace de l'historique de congestion [39], aujourd'hui battu par notamment *NTHU-route* [61].

Un autre domaine dans lequel les algorithmes de routage sont très étudiés est le routage des paquets tels les routeurs réseau pour les applications *TCP/IP*, la qualité de service, réseaux sans fil, traffic sur réseau routier, etc. Bien que la problématique soit sensiblement différente, notamment l'aspect routage dynamique et évolutif du réseau, le cadre théorique dans le domaine de la qualité de servise (*QoS*) a été nettement plus recherché et la littérature très nombreuse depuis le début des années 1990, cinq ans avant les publications de *VPR* et *PathFinder*. Une formulation couramment utilisée aujourd'hui dans ce domaine est notée « » ou « *Delay-constraint least cost problem* » (DCLC).

La littérature en rapport avec la *QoS* décrit de nombreux algorithmes pour attaquer ce problème prouvé NP-difficile [45] lorsque l'expression du coût d'un lien, $c(l)$, est la somme de 2 ou plusieurs métriques. La gestion de la qualité de service demande de tenir compte de nombreux paramètres, tels que le nombre de sauts, la bande passante, le délai, la probabilité de perte de paquets, etc. Cependant, il est également connu que le problème du plus court chemin associé à une seule contrainte est NP-complet, comme analysé dans [71].

On dénote deux types d'approches pour le DCLC : la première effectue une recherche pour chaque métrique, et propose une solution pour mixer les différentes solutions entre elles. La deuxième calcule une métrique comme une combinaison (linéaire ou non) des métriques de coût et de délai, et applique un algorithme de recherche de plus court chemin. La première catégorie fut historiquement la première étudiée. Une approche optimale et naïve est décrite dans [72], et consiste à énumérer l'ensemble des contraintes sur l'ensemble des chemins possibles. Sa complexité en pire cas est exponentielle, et donc non praticable pour des problèmes de taille significative tels qu'adressés dans cette thèse. Une approche « gestion d'échec par règle » (*rule-based fallback*) est décrite dans [44] : une route la plus courte est calculée selon une première contrainte. Si les autres contraintes ne sont pas respectées, alors la route est re-calculée selon une autre contrainte. Cette approche est facile à implémenter, mais est aujourd'hui dépassée par des algorithmes comme LARAC citée dans la section précédente.

Dans la deuxième catégorie d'algorithmes utilisant une approche séquentielle, une proposition de combinaison linéaire de plusieurs contraintes en une seule est faite dans [73]. Les auteurs utilisent cette approche pour calculer les extrêmes de chaque contrainte, mais laissent inexplorés les chemins intermédiaires. Les mêmes auteurs proposent dans [8] une amélioration utilisant l'algorithme de Dijkstra (plutôt que Bellman-Ford), capable d'explorer ces chemins intermédiaires. Cet algorithme, nommé $QoSR_{DKS}$, calcule les extrêmes pour chaque contrainte, mais est capable de construire une combinaison des différentes solutions pour en construire une meilleure. Cette solution exige cependant la combinaison préalable des différentes contraintes en une seule, réduisant la complexité du problème à la recherche du plus court chemin et du chemin le moins coûteux. De manière générale, la combinaison linéaire de plusieurs contraintes est séduisante et peut être efficace pour un problème particulier, mais rend l'algorithme très dépendant des poids choisis pour la combinaison linéaire des différentes contraintes [44]. Ces techniques sont applicables au *WaferIC*, cependant représentent un surcoût en terme de complexité de calcul face aux techniques de routage développées dans cette thèse. En effet, l'attrait de ces techniques est également dans leur généricité : elles sont applicables à tout graphe. Le *WaferIC*

ayant un réseau régulier plutôt dense en comparaison aux *FPGA*, *VLSI* ou *PCB*, il est possible d'utiliser des approches s'appuyant sur cette régularité pour réduire la complexité, et donc les temps de calcul.

2.1.2 PathFinder pour FPGA : une formulation identique au DCLC

Cette section montre que la formulation du problème du « *Delay-constraint least cost problem* » (*DCLC*) utilisé par les algorithmes de routage par paquet (*QoS*) est identique à l'approche des premiers algorithmes performants pour *FPGA*, les routeurs orientés gestion de congestion. L'intérêt est de faire le rapprochement entre ces deux mondes. Le premier est fondé sur une description des problèmes par l'exemple (et a mené à d'excellentes heuristiques), le deuxième sur une description mathématique qui a apporté des algorithmes plus génériques. Ainsi, les apports de l'un et de l'autre permettront de mieux comprendre le défi de ce genre de routage. L'algorithme développé dans cette thèse se base sur ces approches.

La formulation mathématique de la problématique de *DCLC* a été élaborée par plusieurs auteurs. Nous utiliserons celle décrite dans [46] pour sa clarté, mais la première formulation théorique a été réalisé dans [45], soit à la même période que les routeurs *FPGA* performants, de type séquentiels (*VPR*, *PathFinder*). Le problème de DCLC se résume à trouver le chemin au coût le plus faible, étant donnée une contrainte de délai. Le réseau d'interconnexions est modélisé par un graphe dirigé $G = (V, E)$ où V représente l'ensemble des nœuds et $E \subset V \times V$ l'ensemble des liens l dirigés entre ces nœuds (arcs). Chaque lien $l \in E$ est caractérisé par un coût $c(l)$ et un délai $d(l)$ tous deux positifs. Étant donné un nœud source $s \in V$ et un nœud destination $d \in V$, et une contrainte de délai positive Δ_d, le problème de DCLC s'exprime par les équations 2.1.2 et 2.1.3, avec 2.1.2 sujet aux contraintes de 2.1.3 :

$$\min_{p \in P(s,d)} = \sum_{l \in p} c(l) \tag{2.1.2}$$

$$\sum_{l \in p} d(l) \leq \Delta_d \tag{2.1.3}$$

où $P(s,d)$ est l'ensemble des chemins reliant s à d et appelé l'*espace des chemins*.

Bien que la formulation utilisée dans l'article décrivant *PathFinder* [64] soit peu rigoureuse, la description de leur approche est équivalente aux équations 2.1.2 et 2.1.3. En premier lieu, le chemin critique est déterminé par une analyse temporelle statique, ce qui donne le délai maximum autorisé à chaque domaine d'horloge (soit la contrainte décrite par l'équation 2.1.3). Les auteurs construisent une fonction coût explicitée par l'équation 2.1.4 pour un nœud n dépendant des paramètres suivants :

- Le délai du chemin d_n entre la source et la destination en passant par le nœud n ;
- L'historique de congestion h_n au nœud n ;
- Le paramètre p_n, qui dépend du nombre de signaux conflictuels pour l'itération courante au nœud n. Son calcul varie d'un algorithme à l'autre mais représente toujours la congestion à l'itération courante.

De plus, les auteurs introduisent une fonction coût plus évoluée (équation 2.1.5) qui priorise chaque route en fonction de la marge de manœuvre (*slack*) disponible par rapport au « chemin critique ». Le chemin critique est défini par celui ayant le plus petit *slack* dans le domaine d'horloge étudié. Le coût d'un *net n* est donné par C_n, qui est exprimé en fonction d'un calcul intermédiaire c_n :

$$c_n = (d_n + h_n) \times p_n \tag{2.1.4}$$

$$C_n = A_{ij} d_n + (1 - A_{ij}) c_n \tag{2.1.5}$$

Le terme A_{ij} est défini par l'équation 2.1.6 comme étant le ratio du *slack* normalisé :

$$A_{ij} = D_{ij} / D_{max} \tag{2.1.6}$$

Le terme D_{ij} est le délai du chemin contenant l'arc qui relie les noeuds (i, j), et D_{max} est le délai du chemin critique. Le terme A_{ij} est donc bien normalisé entre 0 et 1.

L'algorithme *PathFinder* cherche à résoudre en priorité la congestion (définie par l'équation 2.1.4) tout en équilibrant la priorité avec les problèmes de congestion (équation 2.1.5). Pour reprendre les termes des formulations utilisées par le problème *DCLC*, l'algorithme minimise une fonction objectif. *PathFinder* ne conserve pas un ensemble d'arbres pour minimiser la fonction objectif, mais une approche de recuit simulé faisant évoluer le terme p_n à chaque itération. Bien que cette différence soit suffisante pour classer cet algorithme dans une section à part, la formulation est identique au *DCLC*. Il serait donc possible d'appliquer les algorithmes de routage *Q*oS au problème de routage *FPGA*.

Certes, il est intuitif de penser qu'une approche spécifique est plus performante qu'une approche plus générique (« *no free lunch* ») mais des publications récentes [34, 38] dans le domaine spécifique des routeurs *FPGA* montrent que l'importance de la minimisation du délai permet de limiter la consommation des ressources, et par là même la congestion. Cette approche plus aggressive que *PathFinder*, est supérieure à celui-ci.

Pour limiter toujours plus la congestion, il est important de viser à restreindre toujours plus l'utilisation de ressources, et les approches construites par la formulation du *DCLC* sont particulièrement orientées vers cet objectif. Un des points majeurs est la conservation d'un ensemble d'arbres solutions pour une même paire (source, destination) décrit par l'équation 2.1.2.

2.1.3 Équilibrage des délais

Dans le domaine des *PCB*, des *ASIC* comme dans les *FPGA*, tout routage est contraint sous plusieurs aspects : l'équilibrage des délais en est un, tout comme l'utilisation des ressources, la consommation du système, la compatibilité électromagnétique … La problématique de l'équilibrage des délais est largement présente dans la littérature. Dans le domaine des algorithmes de routage statiques, le problème le plus traité est sans conteste celui des longs chemins critiques [74].

En électronique, un long chemin critique est un chemin dont le temps de propagation de la source à la destination constitue une limite aux objectifs de performance temporelle (ou fréquence d'opération) du circuit. Il est important de comprendre que l'équilibrage des délais est bien plus complexe que la gestion des chemins courts. La gestion des chemins courts en *VLSI* a été prise en compte très tôt, dès les années 1960, au contraire de l'équilibrage de délais. Cette section est focalisée sur la problématique de l'équilibrage, et non pas de la gestion des chemins courts, assez simple car il s'agit d'insérer des délais sur certains chemins, indépendamment les uns des autres. La revue de littérature de la section 2.1.1.2 montre que les algorithmes cherchent un équilibre entre le temps de propagation (à minimiser) et la congestion. Ces approches sont focalisées sur la gestion de contraintes de délai maximum.

Depuis les 10 dernières années, on remarque un certain intérêt pour la gestion des contraintes maximales et minimales, à savoir la gestion des chemins courts et longs. Le premier article publié est certainement [74], un algorithme d'équilibrage des longueurs pour les *PCB*. Focalisé sur la problématique d'une couche dans le cadre de connexions à deux terminaux non-croisées [75] de type bus, les auteurs proposent une solution basée sur le multiplicateur de Lagrange associée à un routage de type séquentiel. L'idée originale et fondatrice est l'assignation de ressources suffisante de routage pour chaque nœud électrique (*net*), assurant de l'espace pour la création d'un serpent. Une analyse statique est réalisée en amont pour connaître les contraintes minimales et maximales de chaque *net* (nœud électrique). Leur approche a été étendue par la suite dans [69] avec une preuve de garantie de qualité de résultat. Cette fois, leur algorithme est basé sur une heuristique pour la réalisation du serpent qui vise à une minimisation de consommation de ressources globale, et démontrée comme optimale.

Le travail présenté dans [74] a été republié dans [76] avec quelques modifications, cependant une véritable extension est proposée dans [77], cette fois basée sur un algorithme de programmation dynamique. De plus, le travail est axé sur la problématique de routage dans le *VLSI* et une comparaison de cette nouvelle approche par rapport à l'originale est faite. Il apparaît que la qualité de routage et d'équilibrage est meilleure avec la nouvelle approche, cependant l'application visée n'est pas la même. Les temps de routage sont par contre nettement plus courts, ce qui tend à montrer que la technique du multiplicateur de Lagrange, quoique très générique, converge lentement dans le cadre de l'équilibrage de délais, et peut osciller entre deux résultats sans converger parfois. De plus, ce travail présente également une comparaison avec une heuristique de recherche qualifiée d'aléatoire, au lieu d'une recherche basée sur la programmation dynamique (PD). Cette approche n'est pas détaillée, mais les résultats sont en terme d'équilibrage aussi bons que la PD, identiques en terme de nombre de coudes, et meilleurs en terme de vias.

Bien que les résultats soient inférieurs en terme de « coût de congestion », un calcul dépendant de l'application et des approches selon les auteurs, ces résultats surprenant ne sont pas discutés. De plus, les temps de calcul ne sont pas rapportés, selon les auteurs, parce que l'approche du multiplicateur de Lagrange utilisée était destinée aux *PCB* [74], et donc toutes comparaisons de temps de calcul est injuste. Il s'agit également d'un bon moyen de ne pas montrer les temps de calcul de l'approche aléatoire ; justement, cette approche aléatoire est intéressante car bien plus simple à

implémenter.

Les résultats obtenus par l'approche proposée dans la présente thèse pour le WaferBoard démontre qu'une recherche faiblement guidée est plus rapide et capable de générer des solutions de qualité quasi-égale à des techniques évoluées et spécifiques. Cette démonstration n'a pas été faite par les auteurs, même dans des publications plus récentes. Quoiqu'il en soit, ces approches sont orientées sur des problématiques de routage hors des réseaux d'interconnexions : entre autres, le réseau utilisé au sein du *WaferIC* ne présente pas un lien entre la distance parcourue et le délai monotone (cf. figure II.2.2).

Quelques auteurs ont repris les idées d'Ozdal *et al* pour les appliquer à d'autres domaines, ou les améliorer. D'autres comme Fung *et al* [34, 38] ont proposé un algorithme sophistiqué appelé *Routing Cost Valley* (*RCV*) pour une allocation du « *slack* » dans le cadre d'un routeur pour *FPGA*. Cette heuristique permet d'évaluer de façon précise les contraintes de délai minimales et maximales pour chaque nœud électrique. Les auteurs ont modifié l'algorithme *PathFinder* pour tenir compte de ces évaluations. Leurs résultats montrent non seulement que de nombreux circuits qui n'étaient pas routables le deviennent, mais également que les réductions de consommation de ressources permettent une légère augmentation des fréquences atteignables sur de nombreux circuits. Leur travaux embrassent une problématique assez proche des besoins du *WaferIC*.

Amouri *et al* [78] ont également proposé un algorithme pour l'équilibrage du délai entre deux *net* : l'algorithme proposé est plus simple, mais moins complet et moins adaptable aux besoins du *WaferIC*. Kubo et al [79] ont étendu le travail d'Ozdal pour réaliser un algorithme d'équilibrage pour *VLSI*, capable de prendre en charge les nœuds électriques avec plus d'un terminal (arbres à router et à équilibrer avec des délais minimum et maximum). Leur travail est basé sur l'analyse géométrique du problème et propose une formulation correspondant à un problème de programmation dynamique. Leur approche permet d'équilibrer des nœuds électriques de degré (fanout) important, mais dans le cadre soit de source-destinations placées sur des lignes parallèles, soit placées sur un rectangle.

En dehors de ces configurations spécifiques, quoique courantes pour les *PCB*, cette approche est inapplicable : en effet les *net* du *WaferIC* à équilibrer ne sont pas nécessairement placés parallèlement les uns aux autres. De plus, la formulation des auteurs assument un réseau maillé simple. Enfin, l'algorithme de routage est séquentiel, et il serait difficile d'utiliser une approche de programmation dynamique pour une partie des *net* seulement.

Plus récemment, un routeur « sans grille » orienté *PCB* [80] a été publié. Cet algorithme résoud la problématique d'équilibrage de façon nettement plus souple ; en effet les autres algorithmes appliquent de fortes contraintes sur les hypothèses, notamment géométriques. Leur algorithme est capable au contraire de router et d'équilibrer des *netlist* dont les sources-destinations ne sont pas situées l'une face à l'autre, et utilisent une approche *Bounded Slice-surface Grid* (BSG) [81] pour contourner des obstacles.

L'approche BSG leur permet d'assigner des zones de routage à chaque nœud électrique pour que cet espace soit utilisé pour l'équilibrage des routes ; la taille des zones est déterminée selon une heuristique résolue par programmation

linéaire. Leur algorithme est probablement le premier, avec *RCV*, à avoir une application industrielle valable. Leur approche est aussi basée sur la programmation linéaire ; celle-ci reste malheureusement difficilement applicable ailleurs que pour les *PCB*, ou au moins les routeurs sans grille, car la formulation du problème et de ses contraintes constituent les entrées du solveur et sont spécifiques à la problématique. Les contraintes de réseau d'interconnexions, pour les *FPGA* comme pour l'application visée, sont fondamentalement différentes.

Sur *FPGA*, les approches basées sur l'allocation du « *slack* » sont utilisées avec plus de succès, tout en proposant des temps de calcul inférieurs. Tout récemment (mai 2011), un algorithme d'équilibrage des délais pour *PCB* [82] a permi de réduire les temps de calcul par rapport au routeur sans grille précédemment cité [80]. Cette approche réalise des équilibrages sur une couche de routage, par l'utilisation de fonctions d'altération de routage (appelés R-flips et C-flips) qui permettent de contourner des obstacles. À chaque obstacle, ces fonctions R et C peuvent être utilisés, et une heuristique permet de déterminer leur sélection. En ce sens, leur approche est similaire à ce que cette thèse propose, le routage par permutations. Cependant, la formulation de leur problème n'est pas directement applicable au *WaferIC*, étant donné que leur fonction de délai est directement proportionnel à la distance parcourue. De plus, la technique de routage par permutation a été développée dans cette thèse en 2009, avant la publication de [82].

Les algorithmes de routage capables d'équilibrer les délais en fonction de contraintes bornées sont récents. Ils couvrent aujourd'hui de manière plus efficace les domaines des *FPGA* et *PCB*, et utilisent soit une approche de programmation dynamique, soit de programmation linéaire. L'approche *RCV* se démarque de par son heuristique et sa solution, très intégrée aux algorithmes de routage *FPGA* et est la raison principale de son utilisation et extension dans cette thèse. La problématique d'équilibrage d'un arbre non-régulier (contrairement aux arbres d'horloge) reste ouverte, bien que dans le cadre très simplifié d'équilibrage entre deux lignes parallèles ou au sein d'un rectangle, Kubo *et al* [79] ait proposé une solution.

2.1.4 Modèle de *netlist*

Dans la communauté de recherche sur les outils CAD, les bancs d'essais fournissent une référence commune pour comparer, caractériser, identifier les limitations et confirmer les capacités d'algorithmes proposés. Pour tout ce qui concerne le routage *VLSI*, trois grands bancs d'essais sont aujourd'hui reconnus. IBM-Place v2 [83], *ISPD'07* et ISPD'08 [54, 13] sont utilisés pour les algorithmes de placement et de routage, et remplacent le vieillissant *MCNC'98* [84]. Côté *FPGA*, le plus utilisé est sans conteste la suite *VPR* [62], réalisée par V. Betz. Par contre, les bancs d'essais publics pour *PCB* sont inexistants : seul *PCB* benchmark 2000 a été publié, mais n'est plus accessible aujourd'hui. Ce problème majeur pour la recherche a été mentionné dans la littérature [85], comme étant la racine du faible nombre de publications dans le domaine, et une entrave sérieuse pour la comparaison d'algorithmes. De plus, la nature confidentielle des *netlist* pour *PCB*, d'autant plus pour les *netlist* placés utilisables pour les algorithmes de routage, empêche les industriels de fournir des *netlists* significatifs. Cette situation ne semble pas évoluer, et c'est donc pour

ces raisons qu'un modèle de *netlist* pour *PCB* a été construit au cours de ces recherches. Un tel modèle est nécessaire pour le routeur du *WaferBoard*, mais également pour la communauté des routeurs *PCB*.

Stroobandt *et al* [86] ont confirmé qu'un générateur de *netlist* configurable permet un controle sur des caractéristiques importantes telles que taille, densité, complexité, fonctionnalités. De plus, les caractéristiques devraient dans l'idéal être indépendantes les unes des autres, et refléter des données réalistes pour l'algorithmes qui les prend. L'objectif des recherches est de construire un modèle de *netlist* pour *PCB*, pour en découler un algorithme de génération et ainsi caractériser des algorithmes de routage. Deux métriques principales sont proposées dans la littérature pour réaliser des générateurs de *netlist* : la loi de Rent [87], et la distribution du degré des *net*. La loi de Rent représente la relation entre le nombre de terminaux (broches) et le nombre de blocs d'un circuit (ou partitions), sous la forme d'une puissance :

$$T = kB^p \tag{2.1.7}$$

ou k représente le nombre moyen de terminaux par bloc et p l'exposant de Rent. Cette loi empirique est en général observée quelque soit la taille de la partition. Il est admis que l'exposant de Rent est compris entre 0.5 et 0.75 [87] dans la région linéaire de Rent. Deux régions supplémentaires sont connues, nommément la région II [87] connue depuis longtemps, et la région III [88]. La région II est située dans la zone de faible nombre de blocs, tandis que la région III se situe parmis les partitions avec de très grands nombre de blocs : la loi de Rent sur-estime le nombre de blocs de cette région. Historiquement, la loi de Rent a été étudiée sur les *PCB* [87] (dans les années 1970), mais les recherches plus récentes se sont focalisées sur *FPGA* et *VLSI*. La loi de Rent s'est avérée notamment un excellent moyen de prédire la longueur des interconnexions entre les blocs après routage, simplement par placement des *netlist* [89, 90]. Cette découverte a permis de grandes avancées pour les algorithmes de placement. La loi de Rent est donc utilisée à la fois pour les algorithmes de placement pour les circuits *VLSI* et *FPGA*, et en même temps pour les algorithmes de génération de *netlist* synthétiques, comme métrique de complexité d'interconnexions.

La seconde métrique très utilisée pour les modèles de *netlist*, est la distribution du degré des *net* (ou fanout). Cette distribution est en général modélisée par une loi exponentielle [91, 92]. La littérature utilise également d'autres modèles, tel que des estimateurs polynomiaux [93]. Une troisième métrique similaire existe également, quoiqu'un peu moins étudiée : la distribution des longueurs des *net* avant routage. Celle-ci n'est pas capturée par la loi de Rent, et permet de modéliser l'étalement géographique des *net*. En général, la distribution des longueurs des *net* est modélisée grâce à une distribution gaussienne [94], et un paramètre de déviation standard. Cependant, un tel modèle n'est pas approprié, comme il sera montré au chapitre 4.4.

2.2 Algorithmes clé de la littérature

Cette section offre une description plus détaillée de certains algorithmes utilisées en totalité ou en partie dans les solutions proposées dans cette thèse. Ces descriptions détaillées sont nécessaires à la compréhension des contributions.

2.2.1 Dijkstra

L'algorithme de Dijkstra a été publié en 1959 [19] et mis au point en 1956. Il est capable de trouver le plus court chemin dans un graphe quelconque, pourvu que le poids des arcs soit positif. Des extensions à cet algorithme permettent aujourd'hui de résoudre le problème quel que soit le poids des arcs.

Le fonctionnement de l'algorithme est le suivant. Soit un graphe $G(V,E)$ avec V nœuds et E arcs, dont les poids sont positifs. Le poids de chaque arc représente le « coût » de franchissement de l'arc, et Dijkstra cherche à minimiser le coût total d'un trajet. Soit v_s le nœud de départ et v_d le nœud de destination. Soit $L(v_i)$ la distance du point de départ à un nœud v_i du graphe en passant par le plus court chemin. Soit V la liste des nœuds visités. Soit $P(v_i)$ les nœuds qui précèdent v_i le long du chemin le plus court.

1. Initialisation : $L(v_s) = 0$ et $V = \emptyset$. Pour chaque nœud v_i du graphe autre que v_s, $L(v_i) = \infty$. Ainsi, il est possible de savoir si un nœud v_i a été visité : si v_i est dans V, alors v_i a été visité ;
2. Soit v_c le nœud courant. Pour chaque voisin immédiat v_{ni} de v_c non visité, calculer la distance d_c nécessaire à l'atteindre en partant de v_s. Mettre à jour $L(v_{ni}) = d_c$ si $d_c < L(v_{ni})$, et marquer $P(v_{ni}) = v_i$. Par exemple, si $L(v_c) = 2$, et le poids de l'arc reliant v_c à v_{ni} est de 8, alors la distance d_c nécessaire pour atteindre v_{ni} en passant par v_c est de $8+2 = 10$. À l'étape 0, $L(v_s) = 0$ (initialisation) ;
3. Marquer v_c comme visité (l'ajouter dans la liste V). Un nœud visité ne le sera plus jamais, son poids est obligatoirement le plus faible dès son premier parcours ;
4. Si le nœud courant est celui de destination, passer à l'étape 5. Sinon, choisir le nœud dont $L(v_i)$ est minimum et qui n'est pas visité, et le marquer comme nœud courant v_c. Passer à l'étape 2 ;
5. Le parcours est terminé : il faut extraire la route finale, chemin le plus court de v_s à v_d. Le parcours s'exécute en sens inverse grâce à la liste P, de la destination à la source. Pour chaque nœud courant v_c (en partant de v_d), il suffit de récupérer $P(v_c)$ pour connaître le nœud précédent. La liste finale est donc à l'envers, pour la connaître à l'endroit il est nécessaire de parcourir la liste P une fois.

La complexité de l'algorithme varie selon l'implémentation. La plus simple et naïve utilise un tableau pour lister les nœuds du graphe, et des tableaux pour les listes L et P ; dans ce cas la complexité algorithmique est $\mathcal{O}(V^2)$. En utilisant des arbres binaires pour lister nœuds et arcs, et une queue à priorité pour L et P, la complexité de l'algorithme est de $\mathcal{O}(V\log(V))$. Malgré tout, le nombre important de nœuds et les coûts cachés dûs à l'implémentation font que Dijkstra est lent pour de grands graphes. Le A* (cf section 2.2.2 le remplace dans ce domaine pour les routeurs dans le

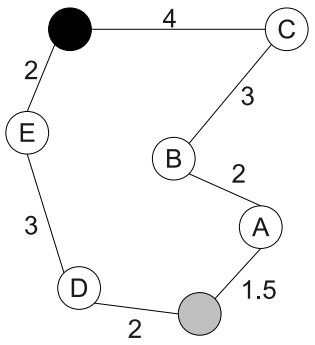
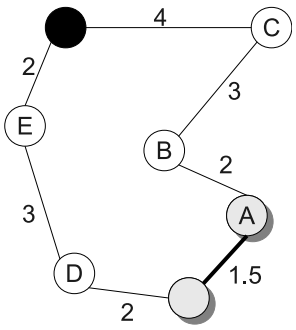
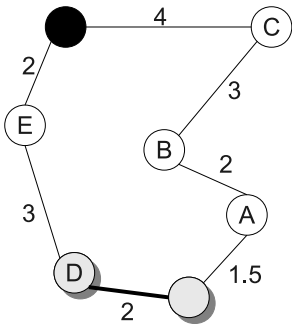


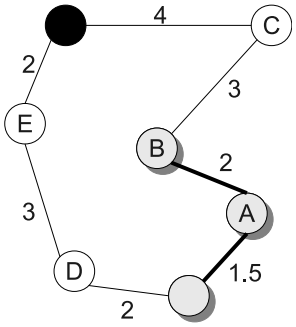
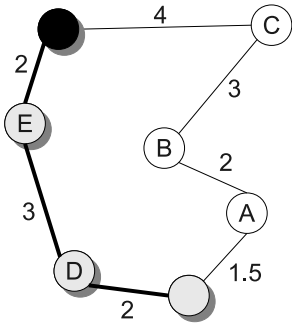
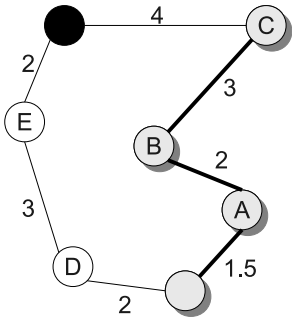
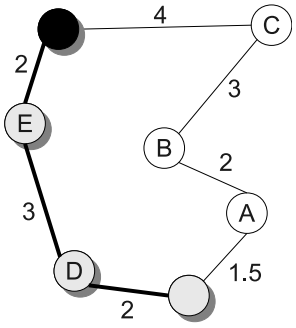
FIGURE 2.2.1: Graphe utilisé comme exemple de résolution du chemin le plus court par l'algorithme Dijkstra et A*. Le nœud de départ est en gris (bas) et le nœud d'arrivée en noir (haut).

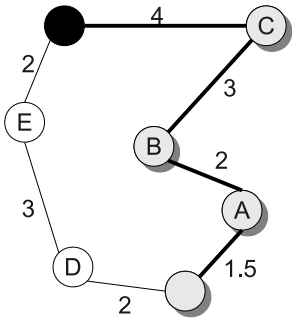
domaine de l'électronique, car les heuristiques de prédictions sont relativement simples à construire pour ces applications.

Pour faciliter la compréhension de cette suite d'instructions, un exemple est donné sur un cas simple. Soit le graphe de la figure 2.2.1 avec comme source le nœud grisé (en bas), et la destination le nœud noir (en haut). Le coût de chaque arc est spécifié le long de celui-ci. Les différentes étapes de l'algorithme sont décrites dans le Tableau 2.3, avec les principales structures de données maintenues par l'algorithme.

Tableau 2.3: Étapes de résolution du chemin le plus court sur un graphe simple, du nœud gris (en bas sans identifiant) au nœud noir (haut) par l'algorithme Dijkstra.

Étape	Figure	v_i	v_{ni}	$P(v_i)$	$L(v_{ni})$	Commentaire
1		v_s	A	-	1.5	L'algorithme commence au nœud de départ, et prend un premier voisin (v_{ni}). Le coût $L(v_{ni})$ est donc de 1.5.
2		v_s	D	-	2	L'algorithme continue l'exploration des voisins de v_s. Le coût $L(v_{ni}) = 2$.

Étape	Figure	v_i	v_{ni}	$P(v_i)$	$L(v_{ni})$	Commentaire
3		A	B	A	3.5	L'algorithme a exploré tous les voisins de v_s ; celui de poids le plus faible est donc A (minimum de la liste L). Le coût du plus court chemin vers B est celui passant par A, et vaut 3.5 ($L(B) = 3.5$.
4		D	E	D	5	Le minimum de la liste L parmi les nœuds non explorés est maintenant le nœud D ($L(D) = 2$). L'algorithme passe donc à l'exploration du nœud E (seul voisin de D). Le coût total de la route la plus courte passant par E est donc 5, et le minimum dans la liste L des nœuds non-explorés est maintenant B.
5		B	C	B	6.5	L'exploration continue du côté de B, avec l'ajout du nœud C. $L(C) = 6.5$, le nœud qui sera étudié par la suite sera donc E ($L(E) = 5$).
6		E	v_d	E	7	L'exploration se fait maintenant sur E, et $L(v_d) = 7$, ce qui est plus important que $L(C)$. L'algorithme va donc étudier pour terminer le nœud C.

Étape	Figure	v_i	v_{ni}	$P(v_i)$	$L(v_{ni})$	Commentaire
7		C	v_d	-	10.5	Le coût total de la route allant à v_d en passant par C est de 10.5, ce qui s'avère plus que celui passant par E. L'algorithme ne met pas à jour $P(v_d)$ et conserve donc E comme prédécesseur. L'algorithme termine donc (avec extraction du chemin si nécessaire, grâce à la liste P).

2.2.2 A*

L'algorithme A* fut décrit pour la première fois en 1968 par P. Hart *et al* [20], comme extension à l'algorithme Dijkstra (1959 par Dijkstra [19]). L'extension principale est l'utilisation d'une liste supplémentaire qui contient une prédiction du coût d'une route passant par un nœud v_c jusqu'au nœud destination v_d. Cette extension réduit le temps d'exécution moyen lorsque l'heuristique est de qualité. Soit un graphe $G(V,E)$ avec V nœuds et E arcs, dont les poids sont positifs. Le poids de chaque arc représente le « coût » de franchissement de l'arc, et A* cherche à minimiser le coût total d'un trajet. Soit v_s le nœud de départ et v_d le nœud de destination. Soit O la liste des nœuds en cours d'exploration (appelée en anglais *open set*), et C la liste des nœuds explorés (appelée en anglais *closed set*).

- Soit $G(v_i)$ le coût de la route la plus courte de v_s à v_i : la liste G correspond à la liste L dans Dijkstra. Il s'agit de coûts connus.
- Soit $H(v_i)$ le coût estimé d'une route partant de v_i et arrivant à v_d. La liste H est donc une estimation.
- Soit $F(v_i)$ le coût total estimé d'une route de v_s à v_d en passant par v_i : à tout moment, $F(v_i) = G(v_i) + H(v_i)$.
- Soit $P(v_i)$ le nœud précédent v_i le long du chemin le plus court.

On définit l'opération $h(v_{i1}, v_{i2})$ comme une fonction de prédiction de coût d'un chemin d'un nœud v_{i1} jusqu'au nœud v_{i2}. Cette fonction donne une estimation du coût d'un parcours, sans connaître à priori s'il existe des obstacles. Une description des étapes de l'algorithme est faite, puis un exemple de cas concret est présenté. L'algorithme se déroule comme suit :

1. Initialisation : la liste initiale des nœuds explorés est vide ($C = \emptyset$), et la liste des nœuds en exploration est initialisée avec le nœud de départ ($O = v_s$). La liste initiale des nœuds précédents est vide ($P = \emptyset$). Concernant la liste des coûts des routes, la liste des coûts connus G est initialisée pour le nœud de départ v_s ($G(v_s) = 0$). Concernant la liste des coûts estimés H, elle est initialisée pour le même nœud ($H(v_s) = h(v_s, v_d)$). Par contre, elle contient une prédiction du coût d'un chemin qui part de v_s et arrive à v_d. Enfin, la liste des coût totaux estimés est mis à jour en conséquence ($F(v_s) = G(v_s) + H(v_s) = 0 + h(v_s, v_d)$). Cette liste permet d'estimer à priori le coût d'une route par rapport à une autre, avant même de router complètement.
2. Si la liste des nœuds en exploration est vide, passer à l'étape 7. Sinon, déclarer le nœud courant v_c : initialiser v_c comme étant le nœud de coût minimum présent dans la liste F (qui contient la liste des coûts totaux estimés pour chaque route testée). À l'itération 0, le nœud courant est donc v_s (seul présent).
3. Si $v_c = v_d$ alors calculer le chemin minimal trouvé. Le chemin minimal est extrait grâce à la liste P exactement de la même façon qu'avec l'algorithme de Dijkstra. Sinon, passer à l'étape 4.
4. Enlever v_c de la liste des nœuds en exploration O, et l'ajouter à la liste des nœuds explorés C.
5. Soit $R =$ faux. Pour chaque nœud v_{ni} voisins immédiats de v_c qui ne sont pas dans l'ensemble des nœuds explorés C : calculer t_g le coût connu de la nouvelle route passant par v_c et v_{ni} ($t_g = G(v_c) + coût(v_c, v_{ni})$). Si le

nœud en cours d'exploration v_{ni} n'est pas encore dans O, l'ajouter et spécifier $R =$ vrai. Si le coût t_g est inférieur au coût connu de v_{ni} ($t_g < G(v_{ni})$) alors spécifier $R =$ vrai.

6. Si $R =$ faux, aller à l'étape 2. Sinon, soit v_{ni} est un nœud nouveau en cours d'exploration, et soit un chemin moins coûteux passant par v_{ni} a été trouvé. Les structures de données sont donc mises à jour comme suit. La liste des prédécesseurs P est mise à jour ($P(v_{ni}) = v_c$). Le coût connu $G(v_{ni})$ est également mis à jour ($G(v_{ni}) = t_g$), et le coût prédit par l'heuristique également ($H(v_{ni}) = h(v_{ni}, v_d)$). Enfin le coût total estimé de la route la plus courte trouvée à ce jour, passant par les nœuds v_c et v_{ni} est mis à jour : $F(v_{ni}) = G(v_{ni}) + H(v_{ni})$. Retourner à l'étape 2.
7. Si l'algorithme se rend ici, alors il n'y a pas de chemin entre v_s et v_d.

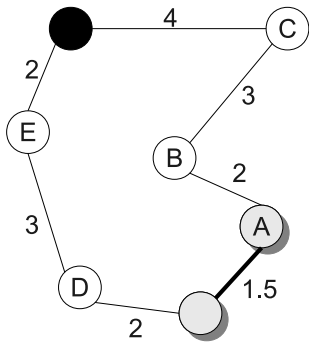
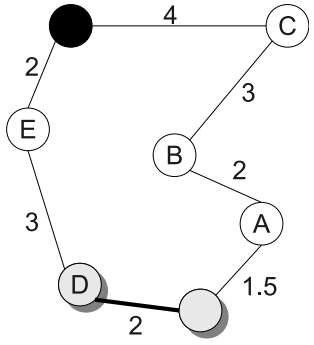
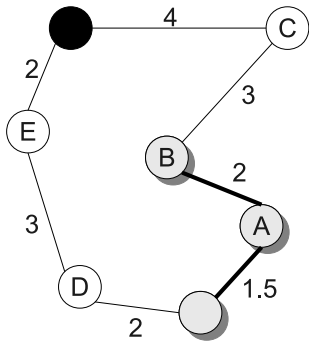
Pour faciliter la compréhension de cette suite d'instructions, l'algorithme est appliqué sur le même graphe que l'exemple précédent. Soit le graphe de la figure 2.2.1 avec comme source le nœud grisé (en bas), et la destination le nœud noir (en haut). Le coût de chaque arc est spécifié le long de celui-ci. Dans cet exemple, l'heuristique de prédiction de coût est la distance à vol d'oiseaux du nœud en exploration jusqu'à la destination. Les différentes étapes de l'algorithme sont décrites dans le Tableau 2.5, avec les principales structures de données maintenues par l'algorithme.

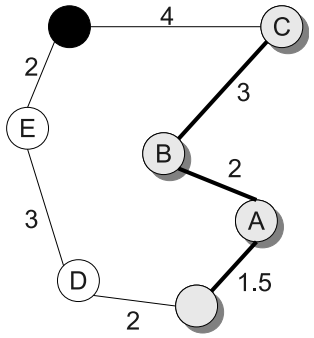
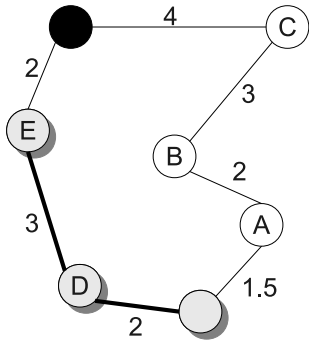
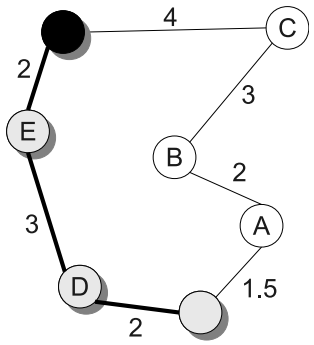
Dans cet exemple, l'algorithme trouve effectivement le plus court chemin. Cependant, une question intéressante mérite d'être posée : que se passe-t-il si le poids de l'arc de C à v_d est de 0 ? À l'étape 4 du Tableau 2.5, l'heuristique de prédiction de coût retournera toujours 4, et donc $H(C) = 10.5$. L'algorithme retournera donc finalement le chemin passant par D et E avec un coût réel total de 7. Or, si le coût de l'arc de C à v_d est de 0, la route passant par A, B et C est de 6.5, soit donc plus petit que de passer par D et E. L'algorithme A* ne serait pas capable de résoudre tous les graphes ? Il s'avère que si, mais que la garantie du plus court chemin n'est présente que lorsque l'heuristique de prédiction ne surestime jamais le coût réel.

Plus clairement, si l'heuristique de prédiction ne donne jamais un coût plus élevé que le réel (quel que soit l'emplacement dans le graphe), alors le A* trouve le chemin le plus court, s'il en existe un. Dans le cas contraire, l'algorithme trouvera un chemin, plus ou moins proche du plus court en fonction de la qualité de l'heuristique, du graphe, de la paire source - destination. Pour une application où la qualité de la solution est importante, l'heuristique se devra d'être admissible. L'heuristique de prédiction peut également prendre en compte des paramètres tels que la congestion locale d'une ressource, ou bien la priorité d'un *net* sur un autre, technique largement utilisée par les algorithmes de routage. Il s'agit là de l'un des intérêts majeurs du A*, et qui lui procure une grande souplesse d'utilisation.

Comparativement à l'algorithme Dijkstra, le nombre d'étapes dans cet exemple est pratiquement le même. En effet, dans cet exemple simple on peut voir que la complexité est la même. En moyenne, la complexité du A* est nettement inférieure à Dijkstra, et en pire cas identique.

Tableau 2.5: Étapes de résolution du chemin le plus court sur un graphe simple, du nœud gris (bas sans identifiant) au nœud noir (haut) par l'algorithme A*.

Étape	Figure	v_c	v_{ni}	$G(v_{ni})$	$H(v_{ni})$	$F(v_{ni})$	Commentaire
1		v_s	A	1.5	4	5.5	L'algorithme explore un premier nœud voisin de v_s : A. Le coût de la route connu est donc de 1.5 (de v_s à A), et le coût estimé (distance à vol d'oiseau de A vers v_d) est de 4. Le coût total estimé est de $1.5+4=5.5$.
2		v_s	D	2	4.5	6.5	L'algorithme continue l'exploration des nœuds adjacents à v_s : D. Le coût de la route de v_s à v_d est de 2, et le coût estimé de D à v_d de 4.5. Au total, le coût estimé de la route v_s, v_d en passant par D est de 6.5.
3		A	B	3.5	2	5.5	L'algorithme ayant terminé à l'étape précédente d'explorer tous les nœuds adjacents à v_s, il choisit de poursuivre l'exploration à partir du meilleur candidat, le nœud A. En effet, $H(A)=5.5$ et $H(D)=6.5$ donc le coût de passer par A est probablement plus faible. On voit bien ici l'intérêt d'une heuristique de qualité. À partir de B, le coût total estimé de la route passant par est de 5.5.

Étape	Figure	v_c	v_{ni}	$G(v_{ni})$	$H(v_{ni})$	$F(v_{ni})$	Commentaire
4		B	C	6.5	4	10.5	À l'étape précédente, le coût total estimé $H(B)$ est de 5.5, et le score suivant est $H(D)$. Ce n'est pas $H(A)$ car ce nœud a été supprimé de la liste des nœuds en exploration (aucun voisin non-explorés). L'algorithme choisit donc de poursuivre l'exploration à partir de B. Le seul voisin est C, et $H(C) = 10.5$. Ainsi, le chemin passant par (A,B,C) semble être plus long que celui passant par D ($H(D) = 6.5$).
5		D	E	5	2	7	L'heuristique de prédiction de distance donne 2 entre E et v_d. Le coût total estimé de la route passant par D et E est de 7 : c'est toujours le chemin le plus court estimé.
6		E	v_d	7	0	7	L'algorithme a trouvé le chemin le plus court [1], son coût est de 7.

1. On ne peut en être certain que si l'heuristique est admissible, ce qui n'est pas le cas dans cet exemple. Cet aspect est discuté dans le texte.

2.2.3 Rip-up and Reroute : retire et recommence

Les algorithmes de routage pour *VLSI* et *FPGA* sont en général de cette famille, appelée en anglais *Rip-up and Reroute* ou *R&R*. Les ressources de routage pour *VLSI* sont régulières et le réseau d'interconnexions simple et non-directionnel. Le principal défi pour le routage est la gestion des congestions. Chaque *net* étant routé un par un (routeur séquentiel), les solutions générées possèdent de nombreux conflits. Pour gérer ces conflits, l'approche *R&R* classique est donc la suivante :

1. Router chaque *net* indépendamment les uns des autres ;
2. Pour chaque *net* conflictuel, l'enlever, puis le rerouter ;
3. Terminer après N itérations, ou s'il n'y a plus de conflit.

La difficulté d'implémentation de ce genre d'algorithme se situe dans l'ordre de routage, et dans la gestion du Rip-up. De très nombreuses variations ont été explorées, mais l'on peut en sortir un schéma général classique. Soit $C(n_i)$ une fonction qui retourne la criticité d'un *net*. Le calcul se fait à partir d'une liste de métriques (typiquement la zone de routage du *net*, sa *slack* [2], ...). Soit $R(n_i)$ une fonction qui détermine si un *net* doit être retirer pour être routé de nouveau. Cette heuristique utilise des paramètres qui varient, mais en général tient compte de la congestion dans la zone de routage du *net*, de sa criticité, du nombre d'itérations réalisées ...

1. Déterminer l'ordre de routage, via l'utilisation d'une heuristique de détermination de la criticité d'un *net* ($C(n)$). Un *net* ayant de fortes contraintes de délai (difficiles à tenir vu la distance, zone traversée ...) sera priorisé, donc routé en premier. Les fonctions de criticité sont assez variables, souvent sous la forme de courbes exponentielles, et prennent en compte un certain nombre de paramètres en fonction du problème visé ;
2. Router chaque *net* dans l'ordre commandé par C ;
3. Déterminer les *net* à retirer, par l'utilisation de l'heuristique R. Chaque *net* est évalué en général : certains algorithmes retirent un grand nombre de *net*, d'autres un faible nombre, et il n'est pas évident aujourd'hui de savoir si une approche est meilleure que l'autre ;
4. Retourner à l'étape 2 s'il y a des *net* à re-router ou qu'un nombre maximum d'itérations ont été effectuées.

La littérature est prolifique concernant ces approches : bien que le concept soit simple au premier abord, déterminer des heuristiques C et R est délicat, fonction des problématiques, et très empiriques. Le routage *R&R* est très largement utilisée pour les routeurs *VLSI*, *FPGA* et *PCB*. Les routeurs pour *FPGA* sont sensiblement différents, notamment après la publication des algorithmes de la famille *PathFinder* et *VPR*.

2. Délai maximal autorisé pour un *net* d'un point à un autre. Ce délai est calculé à partir de la période visée (horloge), du routage et de paramètres de technologies (notamment setup et hold time).

2.2.4 PathFinder et VPR

Les algorithmes *PathFinder* et *VPR* sont très similaires : ils ont été une avancée majeure des algorithmes de routage pour *FPGA* aux alentours des années 1995. *VPR* est toujours développé par l'auteur original (V. Betz) de façon académique, tandis que *PathFinder* a été intégré dans la suite d'outils d'Altera. Les deux algorithmes sont suffisamment proches pour que la description générale de l'approche de *PathFinder* (le premier publié) soit complète : cette section parle donc de l'approche de *PathFinder*, mais est tout à fait similaire à plusieurs dizaines de variations publiées par la suite, dont le très connu *VPR*. La structure générale de l'algorithme réalisé dans cette thèse suit celle de *PathFinder*.

L'algorithme se décompose en deux modules : le premier est un routeur de type A^*, capable de trouver le chemin de coût le plus faible pour une paire source - destination. Le deuxième est un algorithme qui modifie le coût des ressources en fonction de la demande à chaque itération, et qui régit l'ordre de routage des *net*. Ce module modifie donc certains paramètres de l'heuristique de prédiction de coût du A^*, ce qui influe sur les chemins choisis pour chaque *net*, à chaque itération. Les auteurs utilisent dans leur publication originale [64] le terme *global router* pour ce module, mais il s'agit plutôt d'une gestion du routage. En effet, un routeur global réalise un routage sur des grilles plus grossières que le routage final (en général pour des problèmes de mémoire). Une des motivations des auteurs est de se débarasser de l'importance de l'ordre de routage. *PathFinder* supprime donc l'ordre de routage (il les route toujours dans le même ordre, sans tenir compte de la criticité des *net*) et reprend globalement les idées du routage *R&R*, mais lui ajoute une gestion du calcul du coût de chaque ressource en fonction de la demande, gestion qui évolue d'une itération à une autre. Cette méthode transfert donc la criticité des *net* dans le coût des ressources de routage pour chaque *net*. L'algorithme reroute la totalité des *net* à chaque itération (bien que *VPR* sur ce point soit différent). Depuis 1995, des variations ont été proposées pour ajouter un premier tri en fonction d'analyses statiques, comme l'approche *R&R*.

Le coût de l'utilisation d'un nœud n particulier lors du routage d'un *net* est maintenu par le second module. Ce coût C_n est fonction de plusieurs métriques, qui sont :

- Le coût intrinsèque au nœud b_n, c'est à dire au coût inscrit dans le graphe ;
- Un paramètre historique h_n qui tient compte de l'historique de congestion de ce nœud dans les itérations précédentes ;
- Un paramètre de congestion courante p_n proportionnel au nombre de *net* utilisant ce nœud, dans l'itération courante ;
- La criticité du *net* par rapport au slack A_{ij}, calculée par rapport à la fréquence d'horloge désirée. Les paramètres i et j réfèrent à l'arc choisi pour atteindre le nœud n (le délai dans un *FPGA* est dominé par la longueur de l'interconnexion choisie).

Au final, le coût total C_n d'un nœud n est déterminé par l'équation $C_n = A_{ij}d_n + (1 - A_{ij}) \times (b_n + h_n) \times p_n$. Dans la publication originale, la méthode de calcul de h_n, b_n et p_n n'est pas décrite.

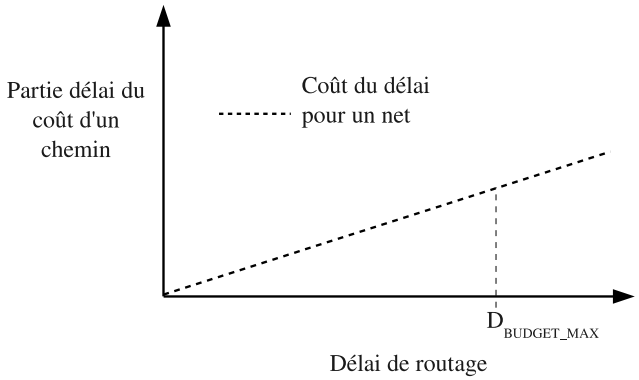


FIGURE 2.2.2: Fonction de coût en rapport avec le délai d'un nœud pour l'algorithme *PathFinder*. La fonction C_n tient compte de cette fonction, qui évolue linéairement.

Les étapes de l'algorithme *PathFinder* (simplifiées pour une meilleure compréhension) sont les suivantes :

1. Déterminer la valeur des paramètres de C_n dépendant du numéro de l'itération ;
2. Pour chaque *net* de la *netlist*, déterminer la liste des paires source - destination (les *net* à grand fanout [3] demandent la construction d'un arbre, appelé Steiner tree [95]) ;
3. Pour chaque *net*, router grâce au A^* en commençant par les paires de nœuds qui sont en bas de l'arbre. Cet ordre permet une meilleure gestion des ressources par rapport à la criticité des délais de chaque *net*. Les paramètres de C_n qui dépendent du nœud choisi lors du calcul d'un chemin sont déterminés dynamiquement ;
4. S'il n'y a plus de conflit, le routage est terminé et réussi. S'il existe des arcs en conflits (plusieurs *net* sur le même arc) et que le nombre maximum d'itérations n'est pas atteint, alors tous les *net* sont retirés. Les coûts liés à l'utilisation de ressources occupées (p_n) sont mis à jour : il devient plus coûteux d'utiliser ces ressources. Cette approche est typique d'un recuit simulé ; à chaque itération, le nombre de *net* en conflit va progressivement réduire car le coût de conflit est augmenté. L'algorithme recommence à l'étape 1.

La partie coût du délai de la fonction C_n est montrée à la figure 2.2.2 : il s'agit d'une fonction linéaire. La simplicité de la description de l'algorithme masque la difficulté à déterminer les valeurs ajustées de chaque variable de l'heuristique. Ces valeurs n'ont jamais été publiées, mais les résultats exceptionnellement bons de l'approche l'ont largement popularisée. Ces idées ont été reprises largement pour les routeurs *VLSI* et de nombreux algorithmes pour *FPGA* : il s'agit aujourd'hui d'une véritable filiation. C'est également sur cet algorithme qu'une extension majeure a été proposée en 2004, puis encore étendue en 2008. Cette extension appelée *Routing Cost Valley* (*RCV*) permet de gérer les contraintes de court chemin, première pierre pour la gestion de l'équilibrage des délais.

3. Par exemple, une source vers deux destinations possède un fanout de 2.

2.2.5 RCV : Routing Cost Valley

Publié en deux parties, la première en 2004 puis avec une extension et sous la forme d'un article de journal en 2008, cet algorithme apporte la gestion des contraintes des courts chemins lors du routage. Il s'agit du premier algorithme publié avec ces capacités pour *FPGA*. Ces contraintes sont un pré-requis pour réaliser un algorithme d'équilibrage des délais et une section de cette thèse prend appui sur *RCV*.

Les principales modifications apportées par *RCV* par rapport aux algorithmes comme *PathFinder* se portent sur le module de gestion des coûts, notamment dans le calcul du délai. Premièrement, *RCV* ajoute le calcul d'un délai objectif D_{target} pour chaque *net* ayant une contrainte de délai minimal. Soit $D_{budgetmin}$ la contrainte de délai minimal pour un *net n*, et $D_{budgetmax}$ le délai maximal autorisé. Ces contraintes peuvent être extraites par une analyse statique de délai (avant routage) et/ou spécifiées par l'utilisateur. Soit D_{min} le délai minimal défini par la technologie utilisée. Les auteurs ne précisent pas un moyen général de le calculer, et utilisent sur *FPGA* 0.1 ns. *RCV* défini le calcul du délai objectif comme suit :

$$D_{target} = min\left(0.5 \times \left(D_{budgetmin}(n) + D_{budgetmax}(n)\right), D_{budgetmin}(n) + D_{min}\right) \tag{2.2.1}$$

Cette équation définie donc que le délai objectif se situe soit à mi-chemin entre le délai minimal et maximal, soit à 0.1 ns au dessus du délai minimum, le plus petit des deux. Le délai objectif est donc au maximum placé 0.1 ns au delà du délai minimum. L'avantage de viser un délai très proche du minimum requis est la minimisation des ressources utilisées, et d'augmenter la marge de manœuvre lors du routage envers la contrainte de délai maximum. En effet cette contrainte est en général nettement plus difficile à rencontrer que la contrainte de délai minimal. La figure 2.2.3 illustre la portion du délai de la fonction de coût lors de l'évaluation d'un chemin.

Pour définir la criticité d'un *net*, ce qui permet de décider lequel aura le plus accès aux ressources, l'algorithme *RCV* défini le calcul suivant :

$$CRIT_{shortpath} = \left(\frac{D_{target}(n) - D_{budgetmin}(n)}{D_{target}}\right)^{\beta} \tag{2.2.2}$$

La variable $CRIT_{shortpath}$ défini le coût d'un *net* en fonction de son délai, par rapport à la contrainte de délai minimum. Le paramètre β (strictement positif) contrôle combien le routeur devrait mettre l'emphase sur la contrainte de délai minimum. Les auteurs utilisent $\beta = 0.5$. La variable $CRIT_{shortpath}$ permet de dessiner la section de la courbe en dessous de D_{target} sur la figure 2.2.3. Le calcul de coût total est une somme de $CRIT_{shortpath}$, $CRIT_{longpath}$ (venant de *PathFinder*) et d'une normalisation.

La gestion de telles contraintes autorise une gestion plus fine des ressources de routage : il a été démontré par les auteurs que de nombreux circuits voyaient leur performance augmenter (fréquences), et le taux d'occupation des ressources de routage diminuer.

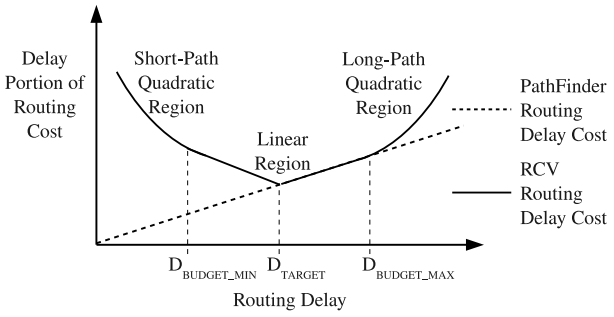


FIGURE 2.2.3: Portion du délai dans la fonction coût lors de l'évaluation d'un chemin pour l'algorithme *RCV*. L'ajout par rapport à la fonction utilisée par *PathFinder* est mise en évidence : une progression exponentielle en dehors de la zone d'intérêt, et un miroir côté chemin court centré sur le délai objectif.

2.2.6 Graphe de Rent

La loi de rent a été explicitée lors de la revue de littérature. Un exemple de graphe de Rent et une méthodologie de calcul de la loi de Rent est présentée ici. Un graphe de Rent est montré à la figure 2.2.4, sur lesquel la région II est visible, mais pas la région III. L'exposant de Rent est calculé selon l'algorithme suivant : pour chaque partition du circuit, le nombre de blocs est calculé (par exemple, le nombre de composants sur un *PCB* ou bien le nombre de slices sur un *FPGA*). Le nombre de terminaux connectés à un autre en dehors de la partition est egalement calculé, et le couple forme un point sur le graphe de Rent. Le nombre de blocs est en abscisse, le nombre de terminaux en ordonnées : sur un graphe log-log, les points s'alignent pour n'importe quelle partition (en dehors des régions II et III).

Le point d'achoppement principal est la méthode de partitionnement. Pour que la loi produise un graphe avec une certaine cohérence, il faut une certaine régularité en fonction de la taille des partitions, quel que soit l'emplacement choisi pour les partitions. Historiquement, le partitionnement est réalisé par subdivisions d'un circuit en zones carrées. Ainsi, la première partition représente la totalité du circuit, le deuxième niveau divise la première en quatre zones égales, le troisième divise chaque partition de niveau deux en quatres sous-zones, etc. Les tailles des zones sont donc des puissances de 2.

Or, cette méthode de calcul présente un biais envers les petites partitions : de très nombreuses petites partitions sont utilisées, ce qui donne un poids important à celles-ci (et donc un poids important à la zone II). Une publication plus récente (2003) a fait l'analyse de ce biais, et recherché des techniques plus évoluées pour éliminer - au moins en partie - ce biais. De toutes celles étudiées, une sort du lot : chaque partition est générée aléatoirement. Le centre de la zone, et sa taille suffisent à construire une zone. Puis, le nombre de blocs et de terminaux est déterminé. Un grand nombre de partitions est généré, ce qui permet de tracer le graphe de Rent. Sur un graphe log-log comme sur la figure 2.2.4, le nombre de points est plus important du côté des grandes partitions, ce qui est normal car sur une echelle linéaire, les points seraient dispersés linéairement.

Une autre technique de partitionnement existe également : les partitions sont déterminées selon une hiérarchie

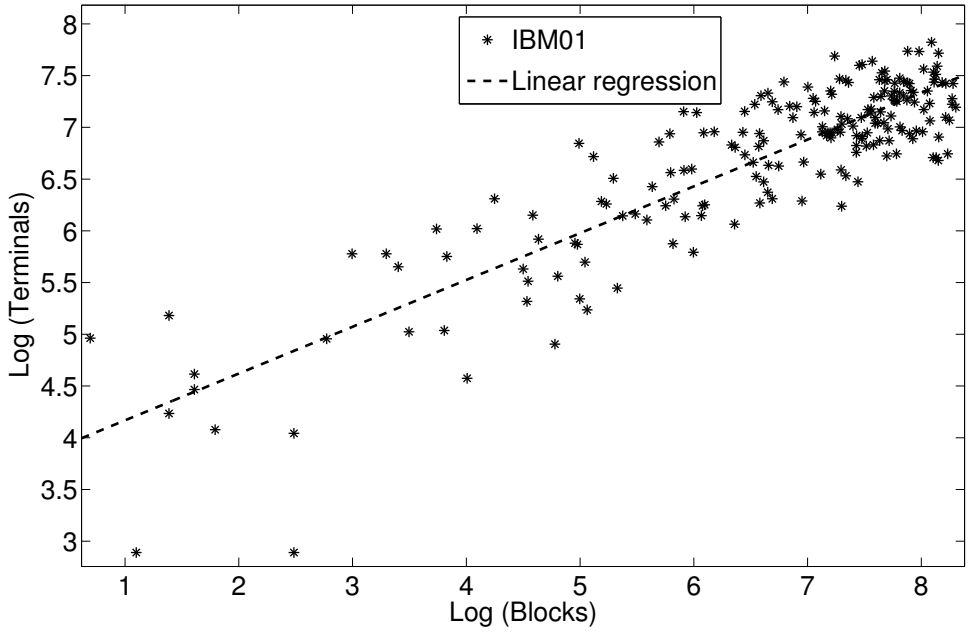


FIGURE 2.2.4: Exemple de graphe de Rent pour le banc d'essai IBM Place 01, calculé en utilisant la technique décrite dans [9]. L'exposant de Rent calculé est 0.45 avec un $R^2 = 0.76$.

existante. Par exemple, sur un *PCB*, chaque composant sera une partition. Sa surface sera rapportée comme le nombre de blocs, et le nombre de broches connectées à des *net* extérieurs comptés comme terminaux.

Quelle que soit la technique utilisée, il convient de garder à l'esprit que cette loi est empirique : elle ne se fonde pas sur une approche théorique bien défini, mais est très utilisée pour prédire la longueur des interconnexions avant routage. Elle sert dans ce cadre de métrique pour quantifier la qualité du placement déterminé par des algorithmes, sur *FPGA* et pour des applications *VLSI*.

Chapitre 3 - Article 1 - A Permutation-Based Routing Algorithm for a Novel Electronic System Prototyping Platform

Étienne Lepercq, Yves Blaquière, Yvon Savaria

IEEE Transaction CAD, soumis le 15 juillet 2012

3.1 Abstract

A recently proposed wafer-sized active integrated circuit capable of programmably interconnecting integrated circuits deposited on its surface needs a routing tool. The desired computation time for this tool is of the order of minutes. In this paper, algorithms for reaching this goal are presented. The first algorithm computes the shortest route between any edge pair for a source-to-destination connection in $O(n)$ (n being the number of edges between the source and destination). It is a non-trivial proposition due to the non-monotonic propagation delay of the interconnection network (IN) used. The second proposed algorithm performs a parallelized random search in the space of same-delay solutions to resolve conflicting routes. The third algorithm solves conflicting routes by searching for detour around conflicting sections, without ripping-up non-conflicting sections.The proposed routing algorithms are characterized with benchmarks for PCB netlists. Our algorithms can route high density netlists (25 % occupancy) on a 80,000 vertices regular IN in about 2 minutes, while typical density netlists (5-15 %) are routed in times ranging from 0.4 to 9 seconds.

routing algorithm, interconnection network, rapid prototyping

3.2 Introduction

An active reconfigurable platform using a Wafer-Scale Integrated Circuit called WaferIC^{TM} was recently proposed for rapid system prototyping by Norman *et al* [96]. Today's electronic system prototyping workflow implies a sequence of iterative steps alternating between PCB development, prototype manufacturing, manual debugging, and PCB fixes.

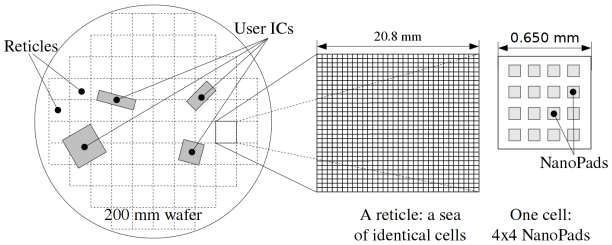


Figure 3.2.1: The DreamWafer board: user's IC are manually deposited on the wafer's surface, external connections are possible as well as probing digital on-wafer interconnection signals – in real time.

One iteration through these steps can take several weeks on complex systems, and multiple iterations may be needed. This has a negative impact on time-to-market and productivity. While Field Programmable Interconnect Chips (FPICs) [10] are capable of reconfigurably interconnecting electronic components, a printed circuit board must be built to match the contact patterns of those components. Furthermore, the limited capacity of FPICs requires the use of multiple chips for the development of high-end circuit boards. This imposes strong limitations on system density and dramatically increases the cost of each prototype.

To overcome these limitations and to significantly reduce electronic system development time, the prototyping platform providing a framework for this research was proposed by Norman *et al*. Further details on the internal structure of the platform were later published in [14]. This prototyping system is based on a wafer-sized CMOS circuit that includes a configurable IN (see schematic figure 5.4.1). This network links a large number of contact points (called NanoPads), each one being configurable as an I/O signal or as a programmable power supply. The NanoPads are grouped by 16 in a simple building block called a cell, where each one supports up to two integrated circuit (IC) pins, i.e. each cell has two "access points", using the terminology used later in this paper. A user simply places integrated circuits on the active surface and an array of NanoPads detects the IC pins. ICs are then programmably interconnected through the defect-tolerant network and the system is ready to run. This step requires a routing algorithm to compute non-conflicting routes through the IN. Signal integrity is maintained by internal repeaters, mitigating crosstalk and attenuation, and ensuring a propagation delay mostly proportional to the wire length. The IN implemented with integrated wires and repeaters introduces delays several times longer than same length PCB copper traces, exacerbating the necessity for an algorithm that focuses on minimizing propagation delays.

User's ICs are powered through the NanoPads, each cell directly supporting several power levels (fed from through wafer vias and tied by suitable power grids). Therefore, the routing algorithm works with PCB-like netlists as input, but with the notable exception that all power and ground nets are not routed. The router uses the predefined IN resources and architecture. This is somewhat similar to FPGA routers linking logic cells using a network architecture different from the one used in our prototyping platform. Most FPGAs use a multi-level architecture (usually, with two levels), as described in [12] and [97] with intra-cluster and inter-cluster architectures, where each logic block connects to communication channels and shares switching blocks resources. In the targeted technology, each cell has access to

routing resources with its own embedded crossbar, connected to different cells both as inputs and outputs.

Besides these differences, the proposed platform and FPGA INs share some common characteristics related to VLSI and PCB routers. Two families of algorithms exist in the literature. Concurrent ones like k-SAT and Integer Linear Programming or those based on Lagrangian Relaxation approaches [98, 7, 2, 42] that are not widely used when short computation times are needed. This is the case when the cost of computing the solution outweighs the benefits derived from an improved solution quality. By contrast, algorithms of the second family are sequential. They are based on problem specific heuristics for PCB, VLSI or FPGA routing. Several FPGA routing algorithms are derivatives of the famous PathFinder [64] and VPR [62, 63] algorithms. They are designed to find a good balance between performance and routability, especially on networks with scarce resources. Sequential algorithms are mainly focused on congestion management heuristics, based on votes and rip-and-reroute techniques. Some algorithms exploit a negotiation-based global router [99] for standard cell design, in addition to a maze router [32] to interconnect source-destination pairs. Our implementation uses the A^* algorithm, which is fast and provides optimal solutions when the heuristic used to estimate distance between source-destination pair is admissible [100] (i.e. when such heuristic never overestimates the distance to the destination). Such approach is common also in VLSI routers such as [101], where sequential routers use iterative rip-up and re-route techniques with penalties cost function based on the congestion history of each edge. Their goal is to minimize total wirelength in very dense benchmarks, like [39, 61, 41], while still providing acceptable computation times. In this paper, the target computation time is expected to be in the order of minutes. This is justified by the set up time and development iteration loop time for the prototyping platform (also called WaferBoard), which is expected to be a few minutes. This contrasts with VLSI development time (measured in weeks or months).

This paper presents the IN and its associated graph representation in section 3.3. In section 3.4, the proposed algorithm called PermFinder is formulated and its important features are analyzed. Section 5.5 is dedicated to our results, which show how the algorithm can trade off propagation delays and computation times, key parameters influencing these goals, and comparisons to the PathFinder algorithm. A conclusion closes the paper.

3.3 The Wafer-Scale Interconnection Network

The IN embedded in the targeted application is a multi-dimensional mesh. The IN needs to cope with generally long interconnects compared to FPGA or VLSI circuits. Since crossing nodes is roughly ten times slower that the wire delay between two nodes in the technology used, long wires avoid this penalty. However since interconnection delays are non-linear in a CMOS circuit for such long connections, repeaters are regularly inserted to achieve a linear delay with distance. This network architecture provides faster paths for long distances than using neighbor-to-neighbor connections, which makes minimum-delay routes not necessarily the shortest in terms of distance. To our knowledge, this non-linear relationship between delay and distance is unique among published FPGA, VLSI and PCB interconnection networks. This contrasts with the PCB or VLSI routing problems that can be modeled with a simple mesh where each

vertex is fully connected to its 4 (2 dimensional mesh) or 6 neighbors (3 dimensional mesh). Global routers for FPGAs use such model because the delay of a path is directly proportional to its length. In the case of the targeted application, the delay model is slightly different, consequently, a new mesh model is needed for the proposed routing algorithm.

The IN is defined as a regular grid of crossbars, where each crossbar is modeled as a vertex in the IN. For each vertex v, let us define a set S_v of distant neighbors in the four directions: the distance between v and its neighbors is expressed by 2^d, $d \in [0;D]$. D is the dimension of the network and is a positive integer. These distances are power-of-two, and an edge is a wire that links v and a vertex in the set S_v. Such construction is depicted in figure 5.4.2, where direct connections (i.e. wires) from the vertex at coordinates (0;0) to its neighbors at distance $2^0, 2^1, 2^2$ are shown. These wires exist for each vertex in the IN. D, the dimension of the network, is sufficient to define the length of the longest wire. For example in the network of figure 5.4.2, $D = 2$ with the longest wire being $\lambda_{max} = 2^D = 4$. Repeaters are placed on long wires to ensure a wire delay being proportional to its length, to compensate for the attenuation in a standard CMOS process. The hardware being implemented, and for which this algorithm is being developed, has currently 6 different lengths of links that correspond to $L = 1$, 2, 4, 8, 16, and 32. Hence the dimension of the network D is 5. This leads to a relatively high wire density that was adopted for two main reasons. First, unlike in FPGAs, VLSI circuits and PCBs, where a smart placer can reduce congestion, our routing tools have no direct control of the user's ICs position on the WaferIC. More importantly, the wafer-scale integration of the hardware sets a requirement for fault-tolerance; crossbars being such key resources for routing, they have been designed to offer abundant resources.

An important property on the delay between two vertices can be derived from the described IN. Let us assume there exists a linear relationship between inter-vertex delay and the distance between a pair of vertices linked by a single edge, which is a reasonable assumption for interconnects with repeaters inserted at regular intervals in CMOS integrated circuits. The total delay of a path depends on (1) the number of vertices from source to destination $n+1$, (2) d_c, the constant delay incurred when crossing each vertex (crossbar), and (3) the wire delay for crossing the distance between two adjacent vertices of distance 1, d_w. Parameter d_w defines the direct proportionality relationship between the inter-vertex delay and the distance between the two connected vertices. In the context of our current implementation of this network, circuit simulations and experimental measurements showed that d_c is about ten times larger than d_w [14]. Note that this ratio depends on the actual size of the cells. It also depends on how the delay through a crossbar compares to wire delays. Therefore, the delay d_u between two vertices linked by one edge is given by equation 3.3.1 where $|\lambda|$ is the length of the edge linking vertices v_s and v_d. The unit length is the cell repetition pitch in the physical network.

$$d_u = |\lambda| d_w + d_c \tag{3.3.1}$$

From equation 3.3.1, the total delay for a path between a source and a destination vertices composed of n edges is

expanded in equation 3.3.2. Some reduction is then applied, and the total delay is expressed by equation 3.3.3:

$$d_p = d_c + \sum_{i=1}^{n} d_{u_i} = d_c + \sum_{i=1}^{n} (|\lambda_i| d_w + d_c) \tag{3.3.2}$$

$$d_p = (n+1)d_c + d_w \sum_{i=1}^{n} |\lambda_i| = (n+1)d_c + d_w \ell \tag{3.3.3}$$

The path delay is proportional to the number of edges on the path and their respective length $|\lambda_i|$. The shortest path delay versus the distance crossed in a multi-dimensional mesh is plotted in figure 3.3.2 for several dimensions of the network. In this paper, λ is the signed length of an edge : it is positive when the edge direction is the same as the v_s, v_d direction, and negative when it is in the v_d, v_s direction. This definition is useful later in the paper.

One important conclusion is that because parameter d_c is much larger than d_w, the delay function is not monotonic with the distance when $D > 0$, a key difference when comparing to FPGA and ASIC routing graph properties. In other words in the multi-dimensional network, it may be faster for a signal to be routed on a somewhat longer path, depending on the distance between the source and the destination. As an example, figure 3.3.2 shows that for $D = 5$ as used in the WaferIC, a delay-optimal route connecting two nodes distant of 32 vertices is taking about 9 ns. On the other hand, another delay-optimal route connecting two nodes distant of 28 vertices is taking about 14 ns. While the distance fly-by distance is smaller by 15 %, the propagation delay is larger by more than 55 %. This impacts the routing model, and explains why the focus of the proposed algorithm is on reducing the delay, and not the distance: they are related, but unlike what is assumed with FPGAs, VLSI circuits or PCBs, the signal delay is not directly proportional to the distance between the source and destination vertices. VLSI routers represent each metal layer with the same topology. Usually, they have their own electrical characteristics, width and preferred directions. Connections between layers are possible as needed, which is not the case for the IN described in this paper. FPGAs share switching resources between vertices (LUTs), unlike the targeted system, where each vertex have access to its own crossbar. VLSI routing algorithms in the literature model each layer as a simple mesh; the IN used in this paper is incompatible with such model, as each "layer" (mesh dimension) has a different network structure. The only common "layer" between the multi-dimensional mesh and a 3D-mesh as used in a VLSI router is the first one. The proposed algorithm takes into account these key constraints. It is of interest that most techniques proposed in this paper could be used in FPGA routers, and probably VLSI routers with some efforts.

3.4 Proposed PermFinder Routing Algorithm

In this paper, a route (or path) is defined as a pair of source-destination vertices. A path is an ordered list of vertices from the source to the destination vertices: several different paths may link the same pair. All edges are unidirectional; two directed edges of opposite directions are available between each vertices pair through the IN. Fault-tolerance is implemented by considering vertices/edges diagnosed faulty as already used. The generic graph representation made for the proposed algorithm can be applied to mesh networks of any dimension. The proposed algorithm, named

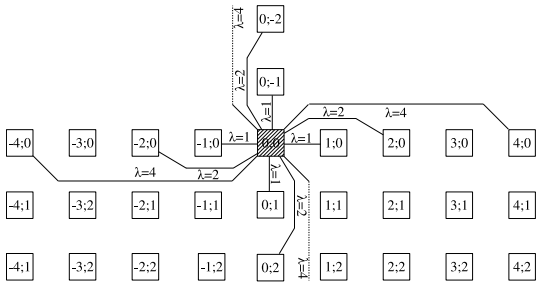


Figure 3.3.1: Multi-dimensional mesh network: vertices are represented by squares (physically, crossbars) with their relative coordinates, and edges (wires). Edges for the vertex at coordinates (0;0) are shown for a mesh of dimension 2 ($2^2 = 4$).

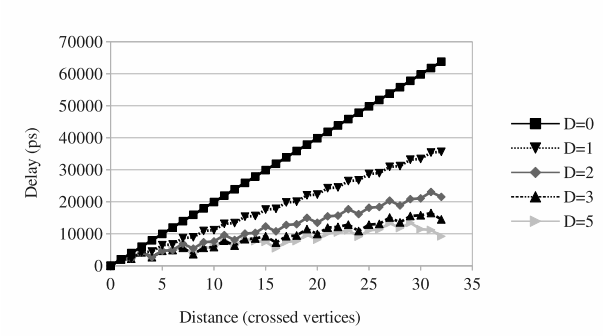


Figure 3.3.2: Delay of the shortest path, function of distance in terms of crossed vertices, for several interconnection network dimension. D=5 is the network dimension in the WaferIC, with $d_c = 10d_w = 2300$ ps. One important property of the interconnection network is that the delay versus distance is not a monotonic function when $D > 0$.

PermFinder, represents the IN as a directed graph. It uses FLUTE [102] for handling Steiner trees, a common solution when low computation times are needed. Steiner trees are generated before routing, and are not reconsidered later by the algorithm. While such approach reduces the search space and can in theory prevent successful routing, results in section 5.5 show that the proposed approach converges successfully for all netlists, even very dense ones.

The routing problem in this paper is formulated as follows: Given N_s nets belonging to set s, compute a route $R_n = [e_1^n, \ldots e_k^n]$ for each net $n \in s$ such that the following objectives are met in order:

1. Have no edge used by different nets, i.e. no conflict in the final solution;
2. Compute a valid solution in less than 10 minutes for dense netlists (20 % average density, as defined in section 5.5);
3. Minimize sum of total delay for netlist $\sum_{n=1}^{N} d_n$;
4. Minimize routing resources consumption (total wirelength) $\sum_{n=1}^{N} \sum_{i=1}^{k} e_i^n$.

As a prototyping platform, the routing algorithm for the WaferIC prioritizes solutions with no conflict (no overflow). As shown in section 5.5, the proposed algorithm always converges with zero conflict. Then, the proposed algorithm focuses on computation times and minimizing propagation delays, and it leverages the high density IN to compute

Algorithm 3.1 Main steps of the proposed routing algorithm

1. Sort nets according to their timing criticality;
2. Route each source/destination pair independently from each other. Each route is computed in $O(n)$ (n being the number of edges between the source and destination vertices). This step produces one shortest path for each route (taken in isolation);
3. Extract conflicting routes, defined as wire segments used by more than one path;
4. Compute as many as e alternative path for each conflicting route, in hope of finding an optimal non-conflicting solution. Alternate solutions are generated by producing permutations of segments composing a previously computed conflicting minimal delay solution;
5. Iteratively solve the last conflicts with an A* algorithm and PathFinder rip-up techniques. Use edge-based routing for faster computations, by leveraging permutations to compute only conflicting portions of routes.

equivalent solutions in terms of propagation delays.

The proposed algorithm is based on PathFinder, a well-known FPGA router. The rip-up technique used is the one described by the PathFinder authors in [64]. Since it is a sequential router, the order in which nets are routed matters: they are first sorted according to their priority (step 1 of algorithm 3.1). In this paper, the following contributions are presented:

– A non-trivial linear-complexity algorithm to compute optimal delay paths for any given pair of nodes in power-of-two multi-dimensional interconnection network. It is proved to compute delay-optimal solutions, and can serve as an admissible heuristic in an A*;

– A permutation approach for solving conflicts, which is optimal in terms of delay, is parallelizable and significantly reduces computation time;

– An "edge-based" routing approach, which is non-optimal but has very low impact on the routing quality and offers significant reduction of computation times.

The algorithm has four main steps as shown in algorithm 3.1. A key parameter of the proposed algorithm is the so-called *effort parameter*, e. This parameter is a bound on the number of permutations computed in step 4 of algorithm 3.1. The impact of its value on the processing time and on the quality of resulting solutions will be characterized in section 5.5. Permutations of equivalent shortest paths, in terms of propagation delays, are fast to compute (linear time with the number of vertices in the path) and are equivalent by construction in terms of propagation delays.

Nowadays, computers have more and more embedded processor cores [103], thus suitably parallelized algorithms can reduce significantly the computation time. It was found that parallelizing the process of generating permutations does not present a major challenge and can provide substantial speedup, as shown later in this paper. Steps 2, 4, and 5 of the algorithm are described in details in sections 3.4.1, 3.4.2, and 3.4.3 respectively.

3.4.1 Finding the shortest path in an ideal mesh

In the first step of Algorithm 3.1, the routing algorithm takes advantage of the regularity and density of the IN embedded in the wafer, to compute each route *independently from each other*. The IN is assumed to be defect-free, and consequently all interconnects are assumed to be available. In this section, the algorithm is proved to provide the shortest path in terms of delay in the IN between any given vertices pair. Therefore it can be used as an admissible heuristic in an A*, in step 4 of algorithm 3.1.

The following demonstration formalizes the proposed solution for finding the shortest path: examples of the situations formalized in the following are shown with figure 3.4.1 and table 3.7. The time complexity of the proposed method is $O(n)$, where n is the number of edges between the source vertex v_s and the destination vertex v_d. The demonstration considers a one dimension graph with edges of power-of-two lengths. A generalization to two orthogonal dimensions is straightforward.

Let G be a directed graph of V vertices and E edges. Each vertex has the same number of edges. Let d_{p_s} be the delay for the shortest path p_s of route r, associated to a source-destination pair $(v_s, v_d) \in V$. Let λ_i be the signed displacement associated with the i^{th} edge (or hop) of p_s (figure 3.4.1), λ_i being positive when displacement is in the same direction as the straight path from v_s to v_d, and negative otherwise. Let us assume there exists a linear relationship between inter-vertex delay and the distance between a pair of vertices linked by a single edge. This is a reasonable assumption for interconnects with repeaters inserted at regular intervals in CMOS integrated circuits. The delay d_u between two vertices linked by one edge is given by equation 3.3.1 where $|\lambda|$ is the length of the edge linking vertices v_s and v_d. The total delay for a path composed of n edges is expressed by equation 3.3.3. Based on these definitions, let us express the shortest path between any vertices pair inside the graph.

Minimum distance path p_{md}: a path in the set of paths P_{md} is built by iteratively choosing the edge for which the destination vertex is the closest to v_d but does *not overshoot* vertex v_d, starting from source vertex v_s. This first path is defined by the list of the edge lengths composing it, also called L_{md}. All permutations of these edge lengths define alternate paths between the same source and destination vertices. All these alternate paths are inserted in P_{md}. These paths *minimize the total distance* between v_s and v_d. Let $d_{p_{\mathrm{md}}}$ be the total delay of each such paths. A path $p_{md} \in P_{\mathrm{md}}$ is built from a list L_{md} of n_{md} edge displacements, $L_{\mathrm{md}} = \{\lambda_{1_{\mathrm{md}}}, \lambda_{2_{\mathrm{md}}}, ..., \lambda_{n_{\mathrm{md}}}\}$. By definition, the sum of edge displacements in a path is equal to its total length $\ell_{\mathrm{md}} = \sum_{i=1}^{n_{\mathrm{md}}} \lambda_{i_{\mathrm{md}}}$, which is also the distance between v_s and v_d when overshoots are not permitted. Note that total length and distance are the same when overshoots are not allowed, hypothesis that will be relaxed later. The construction of a path $p_{md} \in P_{\mathrm{md}}$ is formally expressed by equation 3.4.1.

$$\forall \lambda_i \in L_{\mathrm{md}}, \{\lambda_i > 0, \nexists \lambda_i^{'} \in [1 : \lambda_i[: \Gamma(v_i, v_d) - \lambda_i^{'} < \Gamma(v_{i+1}, v_d)\} \quad (3.4.1)$$

in which v_i is the vertex reached after the i^{th} edge, v_{i+1} the one reached from v_i after choosing an additional edge displacement $\lambda_i^{'}$ and $\Gamma(v_i, v_d)$ is the distance between an intermediate vertex v_i and v_d.

Let us now consider that a path may overshoot the destination vertex.

Minimum number of crossed edges path p_{me}: a path in the set of paths P_{me} is built by iteratively choosing an edge for which the immediate destination vertex is the closest to the final destination vertex v_d, starting from source vertex v_s. The selected immediate destination vertex is allowed to overshoot the final destination v_d. This first path is defined by the list of the edge displacements that compose it, also called L_{me} . All permutations of these edge displacements define alternate paths between the same source and destination vertices. Let $d_{p_{\text{me}}}$ be the total delay of all paths in P_{me}. These paths *minimize the number of edges* between v_s and v_d. A path $p_{\text{me}} \in P_{\text{me}}$ is built from a list L_{me} of n_{me} edge displacements, $L_{\text{me}} = \{\lambda_{1_{\text{me}}}, \lambda_{2_{\text{me}}}, ..., \lambda_{n_{\text{me}}}\}$ and, as for P_{md}, $\ell_{\text{me}} = \sum_{i=1}^{n_{\text{me}}} \lambda_i$ and $\ell \geq 0 \, \forall (v_s, v_d)$. Notice that not all edges displacements in L_{me} are positive (overshooting is allowed which leads to negative edge displacements), and that paths of P_{me} link the same vertices pair as paths in P_{md}. Therefore, $\ell_{\text{md}} = \ell_{\text{me}} = \ell$. Contrary to paths in P_{md}, for each vertex, the chosen edge displacement λ_{me} in a path $p_{\text{me}} \in P_{\text{me}}$ is the one approaching it the most from the destination vertex v_d as expressed in equation 3.4.2. Thus all paths in P_{me} that are obtained from all possible permutations of edge displacements in L_{me} use the minimum possible number of edges.

A formalization of the construction described above is given. Let Λ_i be the set of displacements available in the network interconnection, at vertex v_i. For each displacement $\lambda_i \in L_{\text{me}}$, displacement of a path p_{me} at vertex v_i, there are no displacement $\lambda_i^{'} \in \Lambda_i$, for which destination vertex v_{i+1} is closer to destination vertex v_d than that of λ_i. Equation 3.4.2 is a mathematical expression of this property:

$$\forall \lambda_i \in L_{\text{me}}, \{\nexists \lambda_i^{'} \in \Lambda_i : \Gamma(v_i, v_d) - \left|\lambda_i^{'}\right| < \Gamma(v_{i+1}, v_d)\} \tag{3.4.2}$$

Here v_i is the vertex reached after the i^{th} edge, v_{i+1} the vertex reached when choosing an edge displacement $\lambda_i^{'}$ from the i^{th} edge and $\Gamma(v_i, v_d)$ the distance between intermediate vertex v_i and v_d.

A path p_s with mimimum delay called $d_{p_{\text{opt}}}$ between any vertex pair $(v_s, v_d) \in V$, can be obtained by taking one path either from the P_{md} set when $d_{p_{\text{md}}} \leq d_{p_{\text{me}}}$, or from the P_{me} set when $d_{p_{\text{me}}} \leq d_{p_{\text{md}}}$. The minimum delay of all possible paths between (v_s, v_d) is given by:

$$d_{p_{\text{opt}}} = \min\left(d_{p_{\text{md}}}, d_{p_{\text{me}}}\right) = \min\Big((n_{\text{md}} + 1)\, d_c + d_w \sum_{i=1}^{n_{\text{md}}} \left|\lambda_{i_{\text{md}}}\right|, (n_{\text{me}} + 1)\, d_c + d_w \sum_{i=1}^{n_{\text{me}}} \left|\lambda_{i_{\text{me}}}\right|\Big) \tag{3.4.3}$$

$$d_{p_{\text{opt}}} = \min\big((n_{\text{md}} + 1) d_c + d_w \mathscr{L}_{\text{md}}, (n_{\text{me}} + 1) d_c + d_w \mathscr{L}_{\text{me}}\big) \tag{3.4.4}$$

Let us assume there exists a path $p_{\min}$ with a total delay $d_{p_{\min}} < \min(d_{p_{\mathrm{md}}}, d_{p_{\mathrm{me}}})$ using $n_{\min}$ edges. Equations 3.4.5 and 3.4.6 express such assumption:

$$(n_{\min}+1)d_c + d_w\mathscr{L}_{\min} < \min\Big((n_{\mathrm{md}}+1)\,d_c + d_w\mathscr{L}_{\mathrm{md}}, \\ (n_{\mathrm{me}}+1)\,d_c + d_w\mathscr{L}_{\mathrm{me}}\Big) \tag{3.4.5}$$

$$\sum_{i=1}^{n_{\min}} \lambda_{i_{\min}} = \sum_{i=1}^{n_{\mathrm{md}}} \lambda_{i_{\mathrm{md}}} = \sum_{i=1}^{n_{\mathrm{me}}} \lambda_{i_{\mathrm{me}}} = \Gamma(v_d, v_s) \tag{3.4.6}$$

Case 1: Let us assume $\min(d_{p_{\mathrm{md}}}, d_{p_{\mathrm{me}}}) = d_{p_{\mathrm{md}}}$. From equation 3.4.1, the sum of edge lengths $\mathscr{L}_{\mathrm{me}}$ of path p_{me} and $\mathscr{L}_{\min}$ of path $p_{\min}$ is larger than the sum of edge lengths $\mathscr{L}_{\mathrm{md}}$ of path p_{md}:

$$\sum_{i=1}^{n_{\min}} \left|\lambda_{i_{\min}}\right| \geqslant \sum_{i=1}^{n_{\mathrm{md}}} \left|\lambda_{i_{\mathrm{md}}}\right| \Leftrightarrow \mathscr{L}_{\min} \geqslant \mathscr{L}_{\mathrm{md}} \tag{3.4.7}$$

$$\sum_{i=1}^{n_{\mathrm{me}}} |\lambda_{i_{\mathrm{me}}}| \geqslant \sum_{i=1}^{n_{\mathrm{md}}} \left|\lambda_{i_{\mathrm{md}}}\right| \Leftrightarrow \mathscr{L}_{\mathrm{me}} \geqslant \mathscr{L}_{\mathrm{md}} \tag{3.4.8}$$

By construction, path p_{me} is the one with the smallest number of edges, therefore $n_{\min} \geqslant n_{\mathrm{me}}$ and $n_{\mathrm{md}} \geqslant n_{\mathrm{me}}$. With equations 3.4.7 and 3.4.8, according to the hypothesis that the total length of p_{md} is smaller than that of p_{me}:

$$d_{p_{\mathrm{me}}} > d_{p_{\mathrm{md}}} \tag{3.4.9}$$

$$\Rightarrow (n_{\mathrm{me}}+1)d_c + d_w\mathscr{L}_{\mathrm{me}} > (n_{\mathrm{md}}+1)d_c + d_w\mathscr{L}_{\mathrm{md}} \tag{3.4.10}$$

$$\Rightarrow (n_{\min}+1)d_c + d_w\mathscr{L}_{\mathrm{me}} > (n_{\mathrm{md}}+1)d_c + d_w\mathscr{L}_{\mathrm{md}} \tag{3.4.11}$$

$$\Rightarrow (n_{\min}+1)d_c + d_w\mathscr{L}_{\min} > (n_{\mathrm{md}}+1)d_c + d_w\mathscr{L}_{\mathrm{md}} \tag{3.4.12}$$

$$\Rightarrow d_{p_{\min}} > d_{p_{\mathrm{md}}} \tag{3.4.13}$$

Equation 3.4.13 states that path $p_{\min}$ has a *larger* delay than path p_{md}, which contradicts hypothesis of case 1. Therefore when path p_{md} has smaller delay than path p_{me} there is no other route shorter in the defined IN.

Case 2: Let us assume $\min(d_{p_{\mathrm{md}}}, d_{p_{\mathrm{me}}}) = d_{p_{\mathrm{me}}}$. According to the definition of paths p_{md} and p_{me}, equations 3.4.7 and 3.4.8 are still valid and: $n_{\min} \geqslant n_{\mathrm{me}} \Rightarrow (n_{\min}+1)d_c \geqslant (n_{\mathrm{me}}+1)d_c$

By construction of paths p_{md} and p_{me}:

$$(n_{\mathrm{md}}+1)d_c + d_w\mathscr{L}_{\mathrm{md}} > (n_{\mathrm{me}}+1)d_c + d_w\mathscr{L}_{\mathrm{me}} \tag{3.4.14}$$

$$\Rightarrow (n_{\mathrm{md}}+1)d_c + d_w\mathscr{L}_{\min} > (n_{\mathrm{me}}+1)d_c + d_w\mathscr{L}_{\mathrm{me}} \tag{3.4.15}$$

$$\Rightarrow (n_{\min}+1)d_c + d_w\mathscr{L}_{\min} \geqslant (n_{\mathrm{me}}+1)d_c + d_w\mathscr{L}_{\mathrm{me}} \tag{3.4.16}$$

$$\Rightarrow d_{p_{\min}} \geqslant d_{p_{\mathrm{me}}} \tag{3.4.17}$$

Equation 3.4.17 implies that $p_{\min}$ is larger than p_{me}: when path p_{me} is the shortest path between p_{md} and p_{me}, there is no other route shorter in the defined IN.

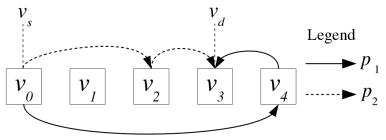


Figure 3.4.1: Two different solutions (with and without overshoot) of a uni-dimensional path between vertices (v_s, v_d) distant of 4 crossbars (vertices). In these examples, path p_2 has the smallest propagation delay (same number of vertices with smaller total distance).

Table 3.7: Examples of paths belonging to P_{me} and P_{md} with vertices pair (v_s, v_d) distant of seven vertices. The interconnection network assumes links of length $\lambda = 1, 2, 4, 8$.

$p_3 \in P_{\text{md}}$ Overshooting path with two edges		$p_4 \in P_{\text{me}}$ Non-overshooting path with three edges	
Edge signed length	Cumulative delay	Edge signed length	Cumulative delay
8	$8d_w + d_c$	4	$4d_w + d_c$
-1	$9d_w + 2d_c$	2	$6d_w + 2d_c$
		1	$7d_w + 3d_c$

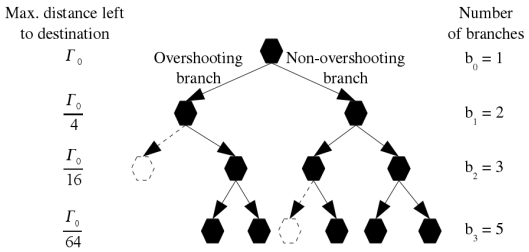


Figure 3.4.2: An example of the decision tree for a route. Each octogon represents a decision point at vertex v, a left-pointing arrow corresponds to a path overshooting at v and a right-pointing arrow corresponds to a path not overshooting at v. Dotted arrows and octogons represent branches that are not explored, in relation to corrolary 3.4.1. The number of branches explored at step j are noted b_i.

Therefore, one can conclude that computing paths p_{md} and p_{me} and keeping the path with the smallest delay between the two is proven to be the path with smallest delay for any edge pair (v_s, v_d) in a regular IN defined by power-of-two vertices length with uniform d_c and d_w.

Examples of paths belonging to P_{me} and P_{md} are depicted in figure 3.4.1 with links of length $\lambda = 1, 2, 4$ and four vertices (crossbars) between v_s and v_d. In this example, path $p_2 \in P_{\text{md}}$ has the smallest propagation delay ($d_{p_2} = 3d_w + 2d_c$), while the delay for path $p_1 \in P_{\text{me}}$ larger ($d_{p_1} = 5d_w + 2d_c$). On the other hand, table 3.7 describes an example of a vertices pair (v_s, v_d) distant of seven vertices with links of length $\lambda = 1, 2, 4, 8$. The total delay of path $p_3 \in P_{\text{md}}$ may be smaller than $p_4 \in P_{\text{me}}$, depending on the value of d_c and d_w. This is true when for example $d_w = 200\,\text{ps}$ and $d_c = 2000\,\text{ps}$ (realistic estimates) where $d_{p_3} = 9d_w + 2d_c = 5800\,\text{ps}$ and $d_{p_4} = 7d_w + 3d_c = 7400\,\text{ps}$.

Computing a path belonging to P_{md} allows for overshooting, and in a multi-dimensional mesh, the selection of

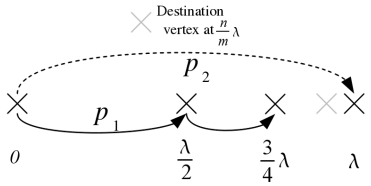


Figure 3.4.3: When destination vertex is in the last quarter of λ (between $\frac{3}{4}\lambda$ and λ), the choice for overshooting may build a path with a smaller delay (theorem 3.4.1).

an overshooting edge may be needed several times for one route. An example of such decision tree is presented in figure 3.4.2. Let b_j be the number of branches (or paths) computed at step j in the decision tree (j denotes its depth). As shown below, the algorithm complexity remains linear with the distance to cross, and the total number of computed paths is b_J, with J the depth of the decision tree depicted in figure 3.4.2. Let us demonstrate that (i) the maximum depth of the decision tree is bounded by J, which depends on the IN architecture, and (ii) that the number of computed branches for a depth of J is finite. From (i) and (ii) the maximum number of computed paths is bounded by constant b_J and algorithm complexity is in $O(b_J\Gamma(v_s,v_d)) = O(\Gamma(v_s,v_d))$

One may note that the edge selection for overshooting appears only if the distance to cross is less than the length $|\lambda_{\max}|$ of the longest edge in the IN. In other words if $\Gamma(v_i,v_d) < |\lambda_{\max}|$. Without loss of generality, figure 3.4.3 depicts the situation when computing the shortest path between two vertices in the graph. Let λ be the displacement associated with an edge, with $\lambda = 2^i\lambda_{min}$, and λ_{min} be the shortest displacement available on the graph, while i is a positive integer.

In a graph with power-of-two edges, the decision of overshooting the destination vertex may build a path with smaller delay if and only if the destination vertex is between $\frac{3}{4}\lambda$ and λ from the current vertex, with λ the displacement associated with the edge subject to that decision.

Let v_s and v_d be respectively the source and destination vertices, and $\lambda_{min} \leq \Gamma(v_s,v_d) \leq \lambda = 2^i\lambda_{min}$

Case 1: let us assume $1 \leq 2^i \leq 2$ (equivalent to $i = 0$), therefore v_d is either at λ_{min} or $2\lambda_{min}$ from v_s. The shortest path (in terms of delay) from v_s to v_d is always the path using one edge, respectively of length λ_{min} or $2\lambda_{min}$.

Case 2: let us assume $4 \leq 2^i \leq \lambda_{max}$ and $\Gamma(v_s,v_d) = \frac{3}{4}\lambda = \frac{3}{4}\times 2^i\lambda_{min}$ the total distance between v_s and v_d. As illustrated in figure 3.4.3, let us build two paths, p_1 that does not overshoot v_d, i.e. that do not use displacement edge λ and p_2 that overshoots v_d by using λ. From v_s to v_d, the total delay of both paths are expressed in the following equations: $d_{p_1} = 3d_c + \frac{3}{4}\lambda$; $d_{p_2} = 3d_c + \frac{5}{4}\lambda$

Therefore $d_{p_1} < d_{p_2}$ when $\Gamma(v_s,v_d) = \frac{3}{4}\lambda$. Consequently, there always exists a path not overshooting the destination that is shorter in terms of delays when $\lambda_{min} \leq \Gamma(v_s,v_d) \leq \frac{3}{4}\times 2^i\lambda_{min}$ with $1 \leq 2^i \leq \lambda_{max}$. The demonstration focuses on the edge at $\frac{3}{4}\lambda$ as if $v_d$is between 0 and $\frac{1}{2}\lambda$ the same analysis holds with a possible overshoot at $\frac{1}{2}\lambda$, whereas in the $\frac{1}{2}\lambda$ to $\frac{3}{4}\lambda$ segment, paths passing through λ necessarily have a delay longer than any competing paths passing path through $\frac{1}{2}\lambda$ that are not overshooting.

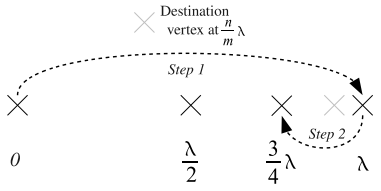


Figure 3.4.4: A path that overshoots its destination twice in a row reaches the edge at $\frac{3}{4}\lambda$. Therefore, it always builds a path with a delay longer than a path passing through $\frac{1}{2}\lambda$ that does not overshoot (theorem 3.4.1).

In a graph with power-of-two edges, overshooting may build a smaller delay path if and only if $\Gamma(v_s, v_d) \geq 6\lambda_{min}$.

From theorem 3.4.1, overshooting may lead to shorter delays if and only if $\frac{3}{4} \times 2^i \lambda_{min} \leq \Gamma(v_s, v_d) \leq 2^i \lambda_{min}$.

It has been also demonstrated that overshooting does not build a shorter path when $i \in [0;2]$. Therefore, overshooting may build a shorter delay path when $i \geq 3$, i.e., when $\Gamma(v_s, v_d) \geq \frac{3}{4} \times 2^3 \lambda_{min} = 6\lambda_{min}$.

In a graph with power-of-two edges, a path with two consecutive edges that overshoot the destination vertex has a longer delay than another one overshooting only once.

As demonstrated in theorem 3.4.1, overshooting can provides a smaller delay path if and only if $\frac{3}{4} \times 2^i \lambda_{min} \leq \Gamma(v_s, v_d) \leq 2^i \lambda_{min}$.

When a path overshooting v_d is built and v_d is in the last quarter of λ (figure 3.4.3), the first displacement edge chosen reaches edge at λ from v_s; overshooting again at step two reaches edge at $\frac{3}{4}\lambda$: from theorem 3.4.1, this builds a path with larger delay than not overshooting it at first. This situation is illustrated by figure 3.4.4.

For each decision of overshooting or not, two paths have to be computed; moreover, this may happen several times for the same path. As demonstrated in the following, the maximum depth J of the decision tree is bounded and this maximum depth depends on the D, the dimension of the IN.

Let us assume the distance between source and destination vertex $\Gamma(v_s, v_d) = \frac{n}{m}\lambda = \frac{n}{m} \times 2^i \lambda_{min}$ with v_d being on the last quarter, which is formalized by the following constraints: $\frac{3}{4} < \frac{n}{m} < 1$, $(m,n) \in \mathbb{N}$, $m \geq 4$, $n \in \left]\frac{3}{4}m; m-1\right]$

With the proposed procedure (see figure 3.4.2), building a path to v_d is done recursively; at step 0, the remaining distance is $\Gamma_0 = \Gamma(v_s; v_d) = \frac{n}{m}\lambda$; at step 1, $\Gamma_1 \leq \frac{\Gamma(v_s, v_d)}{4}$; the following equation expresses the recursive function of maximum possible remaining distance:

$$\Gamma_0 = \frac{n}{m}\lambda \,; \Gamma_1 = \frac{n}{4m}\lambda \,; \Gamma_2 = \frac{n}{16m}\lambda \,; \Rightarrow \Gamma_j = \frac{1}{4^j} \times \frac{n}{m}\lambda \tag{3.4.18}$$

The last decision for overshooting happens when $\Gamma_j \leq \frac{1}{4}\lambda_{max}$ (equation 3.4.18). From corollary 3.4.1, the last possible relevant overshooting decision happens when $6\lambda_{min} < \frac{1}{4^j} \times \frac{n}{m}\lambda < 2^3 \lambda_{min}$. Therefore, the maximum number of decision J is bounded with the inequality: $\frac{m}{6n\lambda_{min}} > \frac{4^{J-1}}{\lambda_{max}} > \frac{m}{8n\lambda_{min}}$. One can express it as following:

$$\frac{ln(m\lambda_{max}) - ln(6n\lambda_{min})}{ln(4)} + 1 > J > \frac{ln(m\lambda_{max}) - ln(8n\lambda_{min})}{ln(4)} + 1 \quad (3.4.19)$$

The maximum depth of the decision tree for a given network architecture with power-of-two edges and a destination vertex position defined by $\frac{n}{m}\lambda_{max}$ is:

$$J = \left\lfloor \frac{ln(m\lambda_{max}) - ln(8n\lambda_{min})}{ln(4)} \right\rfloor + 1 \quad (3.4.20)$$

At step j in the decision tree, there are a maximum of b_j branches to compute, with $b_j = \mathscr{F}(j-1)$, $j \geq 2$, $\mathscr{F}(j)$ being the Fibonacci suite.

At step j in the decision tree, there are at least 1 branch per vertex (i.e. octogon in the example of figure 3.4.3). There is an additional branch (overshooting branch) when the previous branch that led to current vertex is not an overshooting branch (corollary 3.4.1). The number of non-overshooting branches at step $j-1$ is equal to the number of vertices at step $j-2$, therefore the number of branches at step j is defined by the recurrence $b_j = b_{j-1} + b_{j-2}$ with $b_0 = 1$, $b_1 = 2$, $j \geq 2$. Such recurrence is known as the Fibonacci suite, except that $\mathscr{F}_0 = 0$, $\mathscr{F}_1 = 1$, $\mathscr{F}_2 = 2, \ldots$ and $b_o = 1$, $b_1 = 2$, $b_2 = 3$.

The well known Fibonnaci suite, once resolved, provides in this context the number of branches to compute at step j. This final solution is:

$$b_j = \mathscr{F}_{j-1} = \frac{1}{\sqrt{5}}\left(\varphi^{j-1} - \varphi'^{j-1}\right) \quad (3.4.21)$$

with $\varphi = \frac{1+\sqrt{5}}{2}$; $\varphi' = -\frac{1}{\varphi}$, $j \geq 2$.

From equation 3.4.20 and 3.4.21, the maximum number of paths that need to be computed on a graph with power-of-two edges is:

$$b_J = \left\lceil \frac{1}{\sqrt{5}}\left(\varphi^{J-1} - \varphi'^{J-1}\right) \right\rceil \quad (3.4.22)$$

Table 3.8 shows the maximum number of paths that may need to be computed for the targeted IN architecture, with various maximum edges lengths. The current hardware architecture supports links with length of up to 32 vertices, therefore a maximum of 3 paths have to be computed. One may note that this result is valid for a one dimension problem. On a 2 dimension surface, a maximum of 6 paths have to be computed (3 paths for each dimension). An important conclusion is that J is constant for a particular network architecture.

The first step of the proposed algorithm computes at most b_J paths in linear time. The algorithm then keeps the one with shortest delay. The algorithm operates by computing each path, as if it was the only one in a defect free network.

Once a path is computed for each route through this first step, a conflict map is extracted from the list of edges associated with each path: an approach similar to most sequential routers. Each non-conflicting path is recorded, while all others are further processed in the next step that computes permutations offering nominally equivalent delays.

3.4.2 Solving Conflicts With Permutations

The IN is integrated at the wafer scale level and the distance between source-destination pairs can be much longer than in a FPGA or a VLSI circuit. A path is defined by a list of edges, in a specific order. One may note that using a different order of the edges list builds a new path: swapping two edges usually creates a different path, each order being a permutation of another. The number of paths that deliver the same delay, belonging to P_{me} or P_{md}, can be very large. Indeed, the number of paths with the same delay is in general $n!$, where n is the number of edges that belong to a path from a source to a destination. An example of two paths belonging to P_{me} with different permutation of edges is illustrated in figure 3.4.5. The proposed algorithm, which focuses on reducing propagation delays, takes advantage of this property to find an alternative path for each conflicting route. Routing by permutations can be looked as a blindly applied pattern, which is used as a conflict-solving technique. It is efficient if testing each pattern (i.e. each permutation) is fast enough to compensate for the randomness of the technique.

Permutations can be computed in different orders, and three strategies were experimented: (1) in lexicographic order (repeatable, sequential order over all possible permutations as described in [104]), (2) one of the longest edge of the path is permuted to all positions first, then using lexicographic order if no solution is found, and (3) in random order until a solution is found, or a maximum number of permutations have been computed.

In the second strategy, there exist n permutations for the longest edge to be placed at all positions for a path with n edges. If no valid permutation is found (i.e. all permutations are conflicting), permutations are then computed in lexicographic order until a valid solution is found or a maximum number of permutations have been computed. The third strategy computes permutations such that position of each edge is selected randomly (uniform distribution); such an approach statistically spreads the edges used in the 2-dimensional space of the graph's vertices. Results reported later demonstrate the superiority of this strategy in terms of computation time.

Permutations can be seen as a mean of generating random solutions for solving conflicts, and are expected to reduce the computation time. However, the best trade-off between computing more permutations and solving conflicts with

Table 3.8: Maximum number of paths computed for a 1-D interconnection network with edges of power-of-two length, for various maximum edge lengths.

Max. edge length	32	64	128	512	4096
Max. decision tree depth	2	2	3	4	5
Max. # computed paths	3	3	3	5	8

a maze router is not known in advance; the proposed algorithm takes a user specified parameter called the effort e defined earlier. Recall that e, the effort parameter, is the maximum number of permutations to be generated in step 3 of Algorithm 1. The trade-offs associated with e are analyzed later in this paper. It is expected that the permutation approach may not be efficient in solving all conflicts in very congested areas, but nevertheless that it will decrease the number of conflicts to be solved by some other methods. In worst case, the complexity for computing permutations for all conflicting routes is in $O(e \times N \times n_i)$, with N the number of conflicting routes and n_i the average number of edges in conflicting paths. Moreover, computation of permutations has been parallelized, by distributing routes to be solved equally on the number of available cores (usually four or more on today's computers). Such parallel implementation slightly increases the efficiency of permutation computation, at a low implementation cost.

The third step of Algorithm 3.1 processes each route for which a non-conflicting permutation was not found. Since computation time is so important in this particular application, an A* algorithm is used. Most sequential routing algorithms use a maze router at some point, and permutations provides an interesting solution for further reducing computation time in the last step, whichever maze router being used. As stated in [105], up to 50 % of the computation time can be spent while using a maze router for solving the last 2 % nets ; reducing the complexity of this step is interesting in several fields of research. The following strategy is proposed to accelerate this last step when the routing algorithm used is slow (typically a maze router), at the cost of a slight increase in total wire length. A cost is associated to each permutation, as follows: let λ_{c_i} be the length of the i^{th} *conflicting edge* of a permutation *perm* and n be the number of *conflicting edges* in the path. The cost c_p of *perm* is defined as the sum of the delays of all conflicting edges: $c_p = \sum_{i=1}^{n} |\lambda_{c_i}|$

This cost is null if *perm* is non conflicting, and is proportional to the total length of conflicting edges. The most interesting path, in terms of computation time when processing remaining conflicts, is when the selected permutation is *the one having the smallest total conflicting length.* Therefore in the routing algorithm, when no non-conflicting permutation is found for a specific route, the permutation with the smallest cost, i.e. the conflicting solution with the smallest total conflicting length, is recorded for the maze router to solve. The advantages of this approach are explained in the following section.

3.4.3 Solving Remaining Conflicts

Let L_c denote the list of paths still conflicting, that have to be processed by some other method such as a maze router. The proposed approach in this section does not require a specific maze router. Optionally, it can leverage the

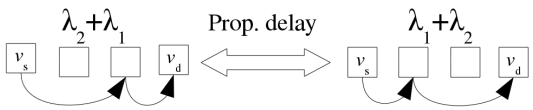


Figure 3.4.5: An example of two equivalent paths with minimum number of edges.

permutation cost information. The impact on computation times and quality of generated solutions is presented in section 3.5.3.

Each conflicting route r_c in L_c is analyzed to extract the list E_c containing the conflicting edges of its associated path p_{s_c}. Each pair (v_{s_i}, v_{d_i}), start and destination vertices of edge $\lambda_{c_i} \in E_c$, is forwarded to the router to find a possibly longer non conflicting route from v_{s_i} to v_{d_i}, as illustrated in the example shown in figure 3.4.6. In (a) the first edge (dotted line) is conflicting with some other route not shown in this figure. A solution (b) is found by the maze router from v_s to v', by routing only conflicting edges. Solution (c) is computed by considering the entire route (source to destination), and is more efficient in terms of propagation delays, but will take more time to compute. In this paper, the solution illustrated in (b) is computed using a technique called "edge based routing" and the one in (c) is called "route based routing". The performance of these techniques are compared in Section 3.5.3.

While this technique can be used without permutations, it can leverage information acquired while computing them. When using the best found permutation (section 3.4.2), the sum of the length of each edge λ_{c_i} in E_c is minimized because p_{s_c} is the permutation which minimizes the cost c_p. This property reduces on average the length of conflicting edges, and further accelerates computation times at later stages. Notice that the cost c_p depends on the effort parameter and the number of edges in a path, and may not be the absolute minimum because not all permutations may have been computed.

If the edge based approach cannot find a solution for some conflicting segment, the associated route is processed later with the route based classic approach. It is also possible that no solution is found by any approach, then iterative rip-up and re-route is used, similar to the approach described in [41].

3.5 Results

Experiments have been performed to characterize the proposed algorithms on several benchmarks created from netlist models, with network sizes between 43,000 and 160,000 vertices. The targeted network size comprises about

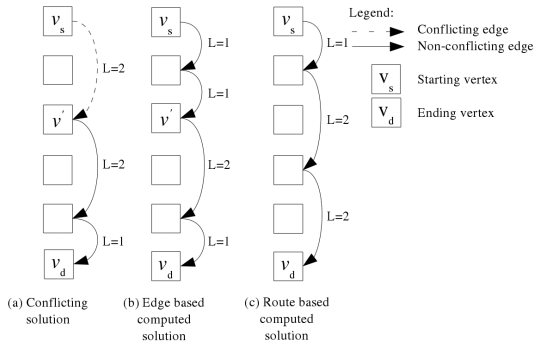


Figure 3.4.6: A conflicting situation example where the dotted edge is used by another route, for a 1-D mesh with links of length 1 and 2 (a). Conflict is solved by routing only conflicting edges (b) or solved by rerouting from start to end (c). The latter ensures the optimality of the solution in terms of propagation delays but takes more time to compute.

80,000 vertices, equivalent to 76 reticle-image times 1024 cells (vertices) per reticle (figure 5.4.1).

Let us recall that the capacity of each vertex is two (two access-points per cell) for the targeted application. The average occupancy rate relates to the *average* number of access points used in each vertex (or IC balls) over the graph. The occupancy rate is called the netlist density in this paper. The occupancy rate locally reaches 100 % (or 2 ICs balls per cell) for netlists with average densities of 20 % and more. First, experiments conducted to calibrate the effort parameter and to compare the various permutation strategies defined in section 3.5.1 are reported. The edge based routing approach is analyzed in section 3.5.3, and the proposed algorithm is then compared to PathFinder in section 3.5.4.

Permutation Strategies

Permutations may be computed using different strategies, and three different were tested. The first one is the lexicographic order discussed earlier, and which is determined by the position of each edge in the route. The second one permutes the longest edge in the path to all possible positions, and then uses a lexicographic order if no solution is found yet. Finally, a random shuffle was tested: the idea of such solution is to spread as much as possible the computed solution over the routing area. Routes at the end of the list (least prioritized routes) may need fewer permutations for finding a valid solution, and the overall computation times would be lowered. To verify this theory, several netlists of various densities were routed using the three strategies: the acceleration factor versus routing without permutations are reported in figure 3.5.1. All these results were obtained with the effort parameter set to 10,000 - that proved to yield a good trade-off between solving conflicts with permutations or with a maze router, as analyzed in the following section. The first two strategies can provide an acceleration of up to a factor 2, on low densities netlist (5 %). On denser netlists, the acceleration is less visible and converges to no speed up. Permutations being an unguided search for non-conflicting solutions, it is expected that the efficiency of such approach is lower on higher density netlist. More interestingly, the random shuffle approach to generate new permutations provides a very substantial acceleration. Random permutations are very efficient on 10 % density netlists for graphs of 82,000 and 100,000 vertices, with acceleration factors of 14.5 and 19. For a smaller graph with 43000 vertices, permutations provide good acceleration too (factors of 3, 6.5, 10.5, 9.5 and 3 for netlists of density 5, 10, 15, 20, 25 % respectively). Finally, for very dense netlists (20-25 %) on graphs of 82,000 to 160,000 vertices the acceleration factor converges to 1, where permutations are not more efficient than a maze router.

3.5.1 Permutations Effort Trade-off

The maximum number of permutations computed by the algorithm is bounded by the effort parameter in the second pass, as explained in Section 3.4.2. Computing all permutations for all routes may take unreasonable time for 10+ edge paths because the time complexity of this step is in $O(n!)$, n being the number of edges of the paths. It is therefore important to find a good trade-off between more permutation tests and solving a difficult route with another algorithm.

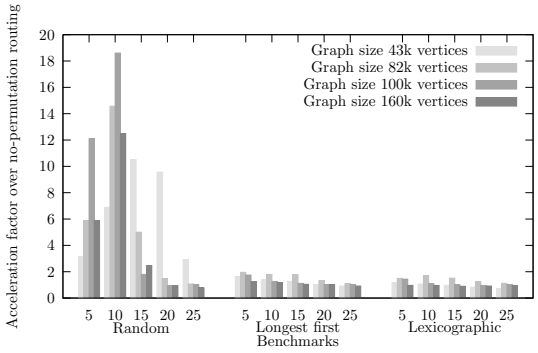


Figure 3.5.1: Acceleration provided by 3 different strategies for permutations, namely random shuffle, longest edge first and lexicographic order (effort of 10,000). The acceleration is computed against routing without permutations. The fastest technique is the random approach on densities of 5-15 %.

Table 3.9: Routing time (in seconds) for various netlist densities and permutation efforts. Results are averaged over 5 runs.

Netlist density	Effort						
	0	1	10	10^2	10^3	10^4	10^5
5	17.0	10.4	7.80	2.02	1.92	1.90	2.44
10	46.8	33.0	27.8	4.51	2.53	2.52	3.33
15	76.5	55.8	48.3	20.4	14.1	18.6	55.3
20	166	149	143	118	119	166	633
25	588	557	554	556	606	694	1689

The best trade-off depends on the cost of computing a permutation versus routing with a maze router, or another algorithm. To characterize this trade-off, an experiment was conducted in which netlists for different network sizes and various densities were routed with several permutation effort values. The routing times in seconds extracted for different scenarios is found in table 3.9 and were measured on a single threaded Intel i7, 2.6 GHz workstation by using the edge-based routing approach.

Depending on the netlist density, the effort parameter has various impacts on the routing time, as shown in table 3.9. Compared to the computation time observed when the effort parameter is set to 10,000, an effort comprised between 0 and 10 increases the total routing time on all benchmarks. For example, the best routing time for a netlist of density 5 % is 1.90 seconds with an effort of 10^4), however using an effort of 0 (no permutation) pushes the routing time to 17 seconds using A*. Because of a lack of space, we did not provide results for other graph sizes, however we have computed them for graph sizes of 40k, 100k and 160k vertices. Other graph sizes and/or denser netlists show a similar pattern, with significant reduction of computation times when using efforts around 1,000 and 10,000 compared to smaller efforts. Small efforts mean that few permutations are tested for each route. With lower computation times when the efforts are around 10,000, it means that it is faster to compute a maximum of 10,000 permutations, reducing the number of routes computed by a maze router compared to experiments with smaller efforts. Using very large

Table 3.10: Percentage of routed nets after *N* computed permutations for various netlists (graph size of 82,944 vertices, effort of 10^6).

	Netlist density	10	20	30
	G (k vertices)	82	82	82
Number of permutations	1	62.65	42.08	30.48
	2	17.88	19.14	15.45
	3-4	11.1	14.02	11.57
	5-8	7.07	15.16	15.05
	9-16	1.00	6.12	10.56
	17-32	0.13	1.97	5.99
	33-64	0.00	0.58	2.97
	65-128	0.06	0.46	2.02
	129-256	0.00	0.08	0.85
	257-512	0.03	0.02	0.29
	513-1024	0.03	0.23	2.55
	>1024	0.03	0.16	2.22
	Not routed by permutations (%)	61.21	39.16	24.56

efforts (usually > 10,000) typically increases the routing time, especially on large networks. It means that the time spent searching for solutions using permutations does not exceed the time it takes to compute the remaining solutions with a maze router. These results are consistent with the distribution of the number of permutations needed to find a valid solution reported in table 3.10. Results in the table correspond to the probability that if a solution is found using permutations, the number of trials to find the first non conflicting solution is the value or a value in the interval reported in the first column. For example, this table shows that for a netlist density of 10 % and a graph size of 82k vertices, if a solution is found by computing permutations, more than 62 % of solutions will be found by the first permutation. Generally, the number of permutations required for finding a non-conflicting solution increases with the density of the benchmark. The number of non-conflicting permutations (valid route) decreases when the number of computed permutations doubles on all benchmarks. However, the probability (in percent) to find non-conflicting routes after 1,000 permutations decreases dramatically, explaining why very large efforts of more than 10,000 are not beneficial in terms of computation times: after 1,000 or 10,000 permutations, it is unlikely to find a valid permutation if one was not already found.

The last row of table 3.10 reports the proportion of routes in percent for which permutations could not find a non-conflicting solution after 10^6 trials. For example, column 1 shows that 61% of routes could not be routed by using the permutation technique, and all these routes were later processed by the A* algorithm. Note that this does not mean that routes were not successfully routed, but gives a clear picture of the success rate of permutations, regardless of the computation time. These results show that permutations are more efficient on large and dense netlists, since the non-routed nets proportion is reducing. At first sight, this result may look counter-intuitive, but recall that the number of possible permutations grows quickly with the number of edges in a route. Indeed, on larger routing area (graph

size) and denser netlists, the proportion of long nets increases on a typical PCB. As a consequence, the number of potential solutions provided by permutations grows rapidly. Moreover, short nets in congested areas have a significant probability of requiring paths longer than the optimal ones: permutations are not performing any search in this solution space. This explains the high percentage of non-routed nets on small graphs.

As a result of the two tables, *using an effort of about 1,000 to 10,000* generally provides the *lowest routing times* on a large netlist characterized by a high density and a large graph size. The general picture of the proposed permutation technique is that it can provide significant improvements in computation times for the WaferIC interconnection network, with no impact on quality of the solution since it explores only path with optimal delays. Such technique shines on a regular architecture in general, and where vertex-to-vertex connections of various lengths exist, as it dramatically increases search space of permutations. Permutations are of general interest, especially for FPGAs which share such characteristics.

For a graph size of about 80,000 vertices, i.e. the full size of the targeted prototyping platform, the proposed algorithm using an effort parameter of 10,000 and the A* for remaining conflicting routes is able to route benchmarks with densities of 5 % in less than 2 seconds on average. A benchmark with an average of 15 % of occupied access points is routed in less than 15 seconds: such benchmarks are considered to be high density netlists when compared to expected usage. The densest tested benchmark occupies an average of 25 % of the access points, and it is still routed in about 10 minutes. This rather unrealistic benchmark represents a netlist with 44,000 pins, which is equivalent to placing 48 of the largest packages (in terms of pin count) offered on the market for FPGAs in the area of a 20 cm wafer. Notice that the WaferIC total area is about 44,000 mm^2. Thus, it could accommodate no more than 22 of those largest pin count FPGAs. All these results that are already satisfactory have been further improved with parallelized permutations, as reported in the following section.

3.5.2 Parallelized permutations

Previous sections have shown several advantages of the proposed random search permutations. This section analyses the effect of the parallelized permutation computation on the execution time. Results have been extracted on an i7 quad core operating at 2.6 GHz with Hyperthreading. It is expected that on very easy netlists, the acceleration factor may reach more than 4 due to HyperThreading, which allows each CPU core to handle 2 threads. As shown in table 3.11, a low netlist density (5 %) shows an acceleration factor of up to 5.29, an excellent result: it allows computation times to decrease from more than 17 seconds (no permutations, A* only) to less than 0.4 seconds using parallelized random permutations with an effort of 1,000. On an high density netlist of 15 %, parallelized permutations offer about 32 % of computation time reduction (with a small 11 seconds computation time) when compared to a single threaded implementation. The advantages of using permutations is smaller on very dense netlists because most of the time is spent executing the A* algorithm, an algorithm much more complex to parallelize efficiently. The parallel approach

Table 3.11: Acceleration factor of the parallel permutations approach (routing time) for several netlist densities and permutation efforts (graph size of 82,944 vertices). The best routing times for each netlist density are shown in italic.

	Permutation efforts					
Netlist dens.	0	1	10	100	1000	10000
5	1 (17)	0.91 (11.4)	1.03 (7.80)	*5.02* (0.40)	5.29 *(0.37)*	4.1 (0.46)
10	1 (46.8)	1.08 (33.0)	1.11 (25.3)	1.86 (2.50)	3.9 *(0.65)*	*3.07* (0.84)
15	1 (76.5)	0.98 (62.0)	1.02 (48.3)	1.07 (20.4)	*1.32* *(10.8)*	1.53 (12.4)
20	1 (166)	1.04 (149)	1.1 (1309)	1.01 (118)	*1.14* *(108)*	1.58 (111)
25	1 (588)	1.03 (557)	1.06 (554)	*1.11* *(505)*	1.1 (550)	1.1 (630)

still demonstrate more than 10 % improvement over single threaded permutations, at 505 seconds of total routing time (less than 9 minutes) for a netlist of 25 % density. Table 3.11 presents all results for efforts comprised between 0 (not using permutations at all) and 100,000. It is of interest that in the experiments reported in Table 3.11, the best value of e is 1000 in most cases and this best observed value of e even goes down to 100 when the netlist density reaches 25%. However, as mentioned before, such high netlist densities are not very realistic and permutations give almost no benefit in those cases.

3.5.3 Edge based versus route based routing

Results reported so far were all obtained with the so-called route based routing. This section investigates the benefits of the optimization called edge based routing proposed in section 3.4.3. Recall that with edge based routing, each edge of a route conflicting with others after computing *e* permutations is processed by a maze router, to solve conflicting segments rather than computing the whole route. This section analyses the impact of this approach on both routing times and quality of the computed solution.

As expected, in some benchmarks, this optimization decreases the routing quality (see figure 3.4.6 and table 3.12). Table 3.12 presents the global routing quality for a targeted graph size of 82,944 vertices, the size of the targeted application. The reported number of conflicts are the one before computing permutations. The number of affected routes measures the number of routes being delayed from the optimal propagation delay. The maximum delay increase is normalized with respect to the delay required for crossing one vertex for the targeted application: normalizing helps measuring the quality of the routing algorithm and not the graph intrinsic performance. Computing permutations can be very effective at preventing conflicts; for example, less than 1 % of routes on all benchmarks need a maze router when the effort is set to 10^6 (recall that an effort between 1,000 and 10,000 is recommended to minimize overall processing

Table 3.12: Impact on route delays for Route based vs. Edge based routing for a graph size of 82944 vertices (effort $e = 10^6$).

Netlist density	# routes	# conflicts before permutations	# affected routes (% over # conflicting routes; % over # routes)		Norm. max. delay increase over d_c (Max. delay increase % over $d_{p_{opt}}$)	
			Route based	Edge based	Route based	Edge based
5	4155	888	0 (0 ; 0)	0 (0 ; 0)	0 (0)	0 (0)
10	8179	3089	0 (0 ; 0)	0 (0 ; 0)	0 (0)	0 (0)
15	12449	6327	1 (0.02 ; 0.01)	5 (0.08 ; 0.04)	1 (2.85)	3 (4.86)
20	16496	10137	3 (0.03 ; 0.1)	6 (0.06 ; 0.04)	1 (2.55)	3 (4.88)
25	20738	14351	20 (0.14 ; 0.1)	36 (0.25 ; 0.17)	2 (6.74)	5 (8.54)

time and in that case about 3 % of the routes need a maze router on average). As expected, the edge based approach induces a slight decrease in routing quality: the number of delayed routes is larger by a factor of about two and each impacted route is longer than with the route based approach (max. delay is larger). However on all benchmarks, the edge based approach computes more than 99 % routes it is requested to handle with their optimal propagation delay. The impacted routes are the 1 % least prioritized ones, while timing constraints are usually applied on 10-20 % of the total number of routes. Since constrained routes are highly prioritized, the impacted routes will always be the least critical ones. The impact of edge based routing on the routing quality is measurable for the targeted application but not significant, even on the largest density benchmarks. Indeed, with each crossbar accounting for about 1.8 ns, in the case of dense netlists (with 20 % occupancy) the worst route would only increase by about 2 crossbars.

The impact on the routing time is shown in figure 3.5.2 as an acceleration factor of the edge based routing compared to the route based approach. On small graphs, route lengths are small and the gain of computing only segments of routes versus computing from start to end, an acceleration factor of about two is achieved. On these benchmarks (graph size of 43,264 vertices), the routing time with both approaches is as small as few seconds. On 100,000+ vertices, route lengths are much larger on average and using the edge based approach is very efficient. As shown in figure 3.5.2, a very important acceleration factor between 10 to 129 is achieved, with nearly no impact on routing quality. In table 3.12 column 4, the number and proportion of non-optimal routes are reported. The computed solution for netlists of density 15 % and more have non-optimal routes, while lower density netlists are routed optimally. As expected, the number and proportion of non-optimal routes grow continuously with the average density, but realistic netlists (up to 20 % density) have a maximum of 0.1 % (route-based) and 0.17 % (edge-based) of their routes with a non-optimal delay. This constitutes a very small number of non-optimal nets: in some rare cases conflicting routes are not solved optimally. The routability provided by the interconnection network of the WaferIC is sufficient and allows for high quality routing. As for wirelength, delay and convergence rates, these metrics are presented in section 3.5.4 and compared with that of PathFinder.

Regarding computation times, the best acceleration is obtained on medium density and large size netlists. The edge

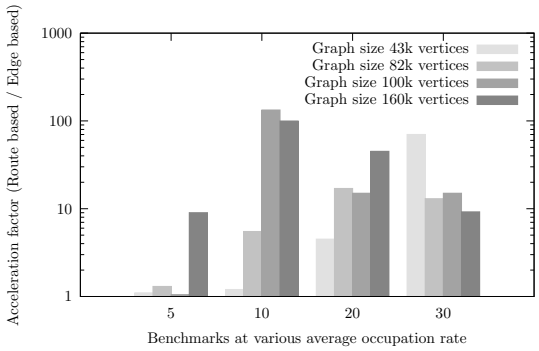


Figure 3.5.2: Acceleration provided by the route based approach versus the edge based approach, for several netlists and graph size and an effort of 10,000. A very important acceleration is seen, especially on large networks with low to medium density netlists. Note that the acceleration factor for graph of 43k vertices on 10 % density netlist is of 1, making it non-visible on the graph.

based routing approach may be useful for routing algorithms looking for computation time reduction at a small cost on routing quality, and it is applicable to most routing algorithms known to the authors.

3.5.4 Comparisons with PathFinder

This section is dedicated to comparing the proposed approach with the PathFinder algorithm, from which our implementation uses the same rip-up and reroute technique. The synthetic benchmarks used for characterizing the proposed algorithm are routed and compared to PathFinder rip-up and reroute. Recall that both algorithms make use of the same A* and algorithm for computing the shortest path presented in section 3.4. This is required as the interconnection network used in the WaferIC is significantly different from ones found in FPGA. Hence, results shown in table 3.13 presents the interest of the permutation approach and edge-based routing. In terms of quality, as expected both approaches provide the same results except for the densest netlist (25 %), where the edge-based approach induces some small impact on total delay and wirelength. Table 3.12 shows that the number of affected routes and overall increase of propagation delay is impacting less than 2 % of the nets. Furthermore, the algorithm sorts nets by priority, such that non-optimal routes are the least prioritized routes. More importantly for the prototyping platform, computation times are clearly much smaller for the proposed approach. Netlists of 5 and 10 % densities are routed in less than a second, an acceleration factor of 46 and 72. Denser netlists are routed in tens of seconds, while an unrealistically dense netlist of 25 % is still routed in less than 2 minutes, at a very low cost regarding quality.

The derived netlists presented so far have interest as they are more closely matching with netlists designed for the WaferIC. However, it is of general interest to look at standard PCB netlists, shown on table 3.14. Such netlists have been designed for PCB, therefore are on average far less dense than the netlists we derived. Except for PCB0 with an average density close to 10 %, all others have average densities of 2 - 5%. All PCBs have between 2 and 14 metal layers (most have 4-6). They comprise from 76 to more than 11,000 nets and from 470 pins to nearly 40,000 component pins. These netlists are routed using the proposed approach with an effort of 1,000 and the edge-based approach, and

Table 3.13: Routing results for PathFinder (R) and the proposed approach (P) using an effort of 1,000 (parallel permutations) and the edge-based routing approach, for a graph of 82,944 vertices.

Netlist density	# routes	# conflicts		Routing time (s)		Total delay sum (ps)		Total wirelength (vertices)		# iterations	
		R	P	R	P	R	P	R	P	R	P
5	4155	0	0	17.0	0.37	$1.16\ 10^8$	$1.16\ 10^8$	$1.02\ 10^6$	$1.02\ 10^6$	1	1
10	8179	0	0	46.8	0.65	$6.28\ 10^8$	$6.28\ 10^8$	$2.02\ 10^6$	$2.02\ 10^6$	2	1
15	12449	0	0	76.5	8.72	$9.41\ 10^8$	$9.41\ 10^8$	$3.04\ 10^6$	$3.04\ 10^6$	2	1
20	16496	0	0	166	21.76	$1.26\ 10^9$	$1.26\ 10^9$	$4.04\ 10^6$	$4.04\ 10^6$	2	1
25	20738	0	0	588	79.40	$1.59\ 10^9$	$1.63\ 10^9$	$5.05\ 10^6$	$5.06\ 10^6$	2	1

Table 3.14: Routing results for PathFinder (R) and the proposed approach (P) using an effort of 1,000 and the edge-based routing approach.

Netlist	# routes	# conflicts		Routing time (s)		Total delay sum (ps)		Total wirelength (vertices)		# iterations	
		R	P	R	P	R	P	R	P	R	P
PCB0	12121	0	0	104.1	10.0	$1.13\ 10^9$	$1.13\ 10^9$	$3.74\ 10^6$	$3.74\ 10^6$	2	1
PCB1	335	0	0	8.1	2.4	$3.52\ 10^8$	$3.52\ 10^8$	$1.15\ 10^6$	$1.15\ 10^6$	2	1
PCB2	2497	0	0	1.1	0.9	$2.40\ 10^7$	$2.4\ 10^7$	75332	75332	1	1
PCB3	2473	0	0	11.3	7.8	$2.40\ 10^8$	$2.40\ 10^8$	804028	804028	1	1
PCB4	1776	0	0	2.6	1.9	$8.55\ 10^7$	$8.55\ 10^7$	271334	271334	1	1
PCB5	687	0	0	1.5	0.9	$2.58\ 10^9$	$2.58\ 10^7$	80797	80799	2	1
PCB6	1268	0	0	2.8	1.8	$8.24\ 10^7$	$8.24\ 10^7$	268293	268293	1	1
PCB7	1674	0	0	2.8	1.9	$1.06\ 10^8$	$1.06\ 10^8$	344913	344913	1	1
PCB8	1823	0	0	4.6	1.9	$1.14\ 10^8$	$1.14\ 10^8$	371089	371089	1	1
PCB9	310	0	0	10.0	8.7	$1.10\ 10^8$	$1.10\ 10^8$	354007	354007	1	1

compared to the PathFinder algorithm.

All netlists of table 3.14 are routed without any conflicts (no overflow) by both algorithms. About 30 % of netlists present some congestion (PCB0, PCB1 and PCB5) and requires rip-and-and reroute to the PathFinder algorithm, while the proposed algorithm manages to route all netlists in one single iteration. On these netlists, the proposed algorithm (P) shows significant improvements in terms of computation times, with acceleration factors comprised between 1.6 to 10 on the most congested and complex netlist. The sum of the total delay and wirelength are the same on both algorithms, or very close to each other (PCB5). Such marginal difference is due to the edge-based routing, which can somewhat lower the final routing quality, but offers significant acceleration over total computation times, as shown in table 3.13. These results confirm that on PCB netlists, which are significantly less congested compared to ones that will target the WaferIC, significant improvements up to a factor 10 can be obtained in terms of computation times, at little to no cost on quality.

3.6 Conclusion

A novel wafer-sized active integrated circuit for rapid prototyping of electronic systems has key constraints that set requirements for a new routing algorithm. Efficient solutions for routing nets in multi-dimensional mesh interconnection networks have been proposed and characterized. The purpose of this algorithm is to focus on very high quality routing to mitigate the high latency of the target network when compared to PCBs, while keeping low routing times for the expected network size and netlist density. A method for computing paths in $O(n)$ has been proposed and proved to provide optimal solutions in terms of propagation delays. A novel technique for handling congestion, computing several routes equivalent to a first solution, or permutations, has been detailed and characterized. This technique is very efficient on netlists of realistic density and graph sizes. It offers both high routing quality and small routing times. It significantly decreases routing times when compared to a straight standard maze router or an A* algorithm for high density netlists, especially when implemented using a parallel approach.

An optimization on the maze router has been proposed. The so-called edge-based routing approach that was proposed and characterized computes conflicting nets only for conflicting edges. Theoretically, this decreases the routing quality, but this paper demonstrated the efficiency of such solution. The total wirelength is impacted by less than a percent, while routing times have been decreased by factors up to 129.

All these techniques have then been directly compared to the PathFinder algorithm on synthetic and PCB netlists. It demonstrated the improvements in terms of computation times, at little to no cost in terms of quality.

Future work will focus on delay balancing over several routes, a must have feature for busses and clock balancing.

Acknowledgments

The authors would like to thank NSERC, Mitacs and Gestion TechnoCap Inc. for financial support, and CMC Microsystems for providing the design tools and access. The authors want to mention HyperChip for its support and for providing one of their industrial-quality PCB project.

Chapitre 4

Équilibrage des délais appliqué au routage de bus

4.1 Introduction

L'intégrité d'un signal au sein du réseau d'interconnexions du *WaferIC* est maintenue par des répéteurs placés à intervalles réguliers. Ainsi, cette technique très connue assure un délai (ou temps) de propagation proportionnel, et non quadratique, à la longueur d'un fil. Cependant, le réseau d'interconnexions introduit malgré tout un délai de plusieurs ordres de grandeur supérieurs à ceux rencontrés sur *PCB*. De plus, le placement des composants est réalisé par l'utilisateur manuellement, dans une optique de prototypage rapide. Le placement n'est donc pas optimal, et des différences importantes de distances (et donc de délai) entre signaux électriques sont attendues. Pour réaliser un système électronique fonctionnel sur *PCB*, il est courant de faire appel à des contraintes de synchronisation, et/ou des contraintes de délai maximum et minimum. De telles contraintes peuvent être définies comme des valeurs absolues (par exemple plus de 5 ns et moins de 7 ns), mais elles peuvent aussi être définie de manière relative par rapport à d'autres. Prenons l'exemple d'un ensemble de signaux électriques appelé bus, utilisé pour une communication *FPGA* / *RAM*. Il est possible de spécifier au routeur utilisé que le délai du groupe (tous les *net* du bus) doit être équilibré, c'est à dire que chaque signal membre du groupe possède un temps de propagation égal aux autres, avec une tolérance. Ce chapitre aborde un tel problème pour des réseaux d'interconnexions réguliers multi-dimensionnels. L'algorithme de routage doit également tenir compte de temps de calcul limités : l'utilisateur du WaferBoard peut mettre en place un système électronique en quelques minutes, en conséquence les temps de calcul acceptables sont réduits à quelques minutes (une dizaine de minutes est visé). Ce chapitre fait l'objet d'un article de journal en préparation, dont la rédaction n'est pas encore assez avancée pour être soumis (cf. annexes).

Les différences d'architectures entre le *WaferIC* et les *FPGA*, *PCB* ou encore circuits *VLSI* ont déjà été abordés à la section 1.1. Les contraintes de routage sont plus proches de ceux sur *FPGA* que sur les autres technologies, et c'est pourquoi le premier algorithme proposé dans ce chapitre pour l'équilibrage de délais est basé sur un routeur bien connu sur *FPGA* [34] : *Routing Cost Valley*, ou *RCV*. La plupart des algorithmes de routage pour *FPGA* sont des dérivés des célèbres algorithmes *PathFinder* [64] et *VPR* [62]. Ceux-ci sont conçus pour trouver un bon équilibre entre

performance et routabilité, en particulier sur les réseaux avec des ressources limitées. Les algorithmes de routage pour le *WaferIC* proposés ici sont basés sur leurs travaux et utilise principalement un A* pour trouver des chemins entre une paire source-destination. Les arbres de routage pour les nœuds électriques avec plusieurs destinations sont pré-traités par l'algorithme *Fast Lookup Table Based Rectilinear* (*FLUTE* [106]) pour trouver le *RSMT* (l'arbre rectiligne minimal de Steiner). L'algorithme traite ensuite les arbres générés sous forme de paires sources-destinations.

La revue de littérature au chapitre 2 présente les publications les plus importantes concernant l'équilibrage des délais. Le lecteur pourra s'y référer pour comprendre le choix de la technique qui a servi d'appui aux algorithmes proposés ici. Fung et al. [34, 38] ont proposé un routeur *FPGA*, appelé *Routing Cost Valley* (*RCV*), qui a ajouté les contraintes de délai court aux routeurs pour *FPGA*. Selon nos connaissances, *RCV* est le plus complet des routeurs pour *FPGA* publié en mesure de satisfaire les deux contraintes que sont les bornes de délais minimum et maximum. Des outils commerciaux en sont capables également, mais selon nos connaissances leur technologie n'a jamais été rendue publique. L'algorithme *RCV* a ajouté aux routeurs antérieurs un algorithme d'attribution de marge de délai (*slack-allocation*) capable de convertir les contraintes de délai basées sur les chemins extraits par analyse temporelle statique, en contraintes définies par un budget de délai pour chaque connexion. Ils ont également établi une fonction de coût pour l'algorithme A * pour diriger la recherche d'un chemin, non plus le plus court, mais qui satisfait des bornes de délais minimal et maximal. Leur algorithme d'allocation de marge de délai peut gérer des connexions en arbre (*fanout* supérieur à 1), et diffuser la marge de délai tout au long de l'arbre. En outre, la fonction de coût proposée par les auteurs est une modification de celle de *PathFinder*, et augmente le coût d'un chemin en cours de routage qui est estimé avoir un délai trop petit (cf. le fonctionnement de l'algorithme A* et *RCV* au chapitre 2). Par conséquent, des contraintes simples de délais minimum et maximum sont gérées. *RCV* utilise les contraintes de délai maximum et minimum pour chaque *net* indépendamment des autres. Leur travail permet de produire des circuits fonctionnant à des fréquences plus élevées et a montré des capacités étendues pour la synchronisation. Cependant, il est montré à la section 4.2.1 que la gestion des délais absolu minimum et maximum n'est pas suffisante pour supporter l'équilibre d'un groupe de taille arbitraire. L'algorithme *RCV* ne gère donc pas efficacement l'équilibre de délais au sein d'un groupe, un élément obligatoire pour l'application visée. Ce chapitre propose une modification de l'algorithme *RCV* d'allocation des délais et y ajoute deux éléments majeurs. Un premier algorithme capable de gérer efficacement les contraintes d'équilibrage dans un groupe, et un deuxième algorithme pour accélérer significativement les calculs, basé sur une table d'équivalences utilisée avec le A*. À ces deux techniques s'ajoute une proposition de modification de la fonction de coût de *RCV*, qui permet d'accélérer très fortement le routage pour l'application visée.

Ce chapitre est structuré comme suit : la section 2 est partagée en trois sous-sections. La première partie décrit l'algorithme d'allocation dynamique des délais. La deuxième propose un algorithme de routage par table d'équivalence. La troisième présente une nouvelle fonction de coût à utiliser par le A*, qui réduit l'espace de recherche sans influer négativement sur la qualité des solutions générées. Les résultats de ces trois propositions sont fournis dans la

section 4.3, et une conclusion résume les principales contributions de ce chapitre.

4.2 Description générale de l'algorithme utilisé

Une route est définie comme une paire de sommets source-destination dans le graphe, représentation du réseau d'interconnexions physique. Un chemin est une liste ordonnée de sommets depuis la source jusqu'à la destination : en général plusieurs chemins différents peuvent établir un lien entre deux sommets. Les sommets représentent les commutateurs programmables du *WaferIC* (*crossbar*, un par cellule), et des arcs dirigés représentent les segments de fil entre chaque cellule. L'algorithme considère les fils et cellules défectueux comme non-utilisables. Ces définitions sont les mêmes qu'utilisés au chapitre 2.2.6, section 3.3. Il a été démontré dans la section 1.1 qu'au sein du *WaferIC*, un chemin p_1 de plus grande longueur que p_2 peut posséder un délai plus petit que p_2. Pour alléger le texte, le terme « chemin le plus court » doit être compris comme « chemin de plus petit délai », et ce quel que soit sa longueur. Les grandes étapes de l'algorithme de routage présenté dans ce chapîitre sont explicitées à la description générale de l'algorithme 4.1.

L'algorithme commence par calculer l'arbre de Steiner minimal pour chaque *net* de degré supérieur à 2 : FLUTE est utilisé pour cela, une implémentation est proposée par les auteurs pour fin de recherche académique. Ensuite, l'algorithme utilise les résultats du chapitre 2.2.6 pour calculer le chemin optimal pour chaque *net*, sans tenir compte de conflits. Le délai minimal possible pour la technologie, que l'on appelle ici optimal, est également enregistré pour la suite.

Chaque *net* ayant été routé sans gestion de conflit, l'étape 3 (cf. algorithme 4.1) consiste à extraire les *net* en conflit : ils sont triés par ordre de priorité, en fonction de leur délai. Il a en effet été montré au chapitre 2.2.6 que cet ordre était plus efficace que l'inverse. Puis, la technique de routage par permutations est ensuite utilisée pour réduire efficacement le nombre de conflits, avant d'entrer dans la section itérative de l'algorithme, à l'étape 5.

L'étape 5 vise à router avec succès les *net* les plus conflictuels, et utilise les techniques de « *ripup and reroute* » développées par *PathFinder*. Ce chapitre détaille les sous-étapes proposées pour gérer efficacement les contraintes d'équilibrage de délai dans un bus (dénommé groupe) de taille arbitraire. Ces étapes se situent au début de l'étape 5, à chaque itération. Tout comme *RCV*, l'équilibrage des délais est réalisé en deux étapes : d'abord par l'établissement des contraintes de délai minimum et maximum pour chaque *net*, puis par le routage avec un A*, qui utilise ces contraintes pour trouver une solution non conflictuelle qui remplit les critères définis à l'étape précédente. *RCV* extrait les contraintes de délai minimum et maximum après une analyse temporelle statique du circuit pour router chaque *net* indépendamment (sans équilibrage au sein d'un groupe). Dans le cadre du *WaferIC* tout comme sur *PCB*, les contraintes proviennent de l'utilisateur mais demandent à être établies relativement les unes par rapport aux autres pour tenir compte de chaque membre du groupe. Il existent des outils commerciaux pour aider à la définition des contraintes, mais celles-ci dépendent de chaque composant utilisé et nécessitent aujourd'hui encore l'intervention de l'utilisateur.

Algorithm 4.1 Étapes générales du routage

1. Traiter chaque *net* de degré supérieur à 2 (arbre) par l'algorithme FLUTE pour calculer le RSMT;
2. Router chaque *net* selon son délai minimal, sans gestion de conflit;
3. Extraire la cartographie des conflits, et trier les chemins conflictuels «plus court en premier»
4. Résoudre les conflits en cherchant dans l'espace des solutions de délai optimal (délai minimal possible au sein du réseau d'interconnexions)
5. Tant qu'il existe des *net* conflictuels, retirer ceux-ci selon les critères utilisés par le routeurs *PathFinder* tel que défini à la section 2.1. Calculer les chemins en utilisant un A*.

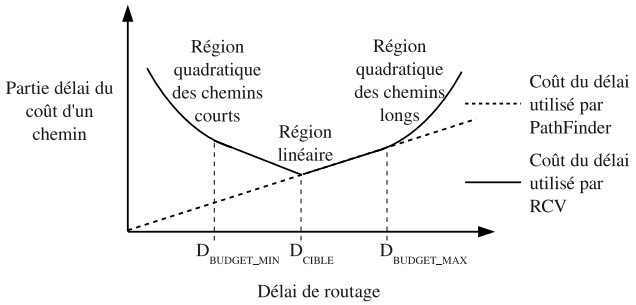


FIGURE 4.2.1: La fonction coût lors de l'évaluation d'un chemin pour l'algorithme *RCV*. L'ajout par rapport à la fonction utilisée par *PathFinder* est mise en évidence : une progression exponentielle en dehors de la zone d'intérêt, et un miroir côté chemin court centré sur le délai objectif.

La section 4.2.1 explicite ce besoin et les solutions proposées pour résoudre ce problème, via un algorithme d'allocation dynamique des contraintes de délai.

4.2.1 Algorithme d'allocation dynamique des délais

Le problème de l'équilibrage de délais pour un groupe de nœuds électriques est formulé comme suit. Soit N une liste de *net* à router, et C la liste des contraintes de délai associée à N. Chaque *net* n est associé à un groupe de *net* s, et aux contraintes c. Les contraintes c spécifient les bornes de délai minimum $d_{\min_n}$ et maximum $d_{\max_n}$ associées au *net* n. Ces constantes sont définies par l'utilisateur et s'appliquent à chaque *net* du groupe. Elles correspondent aux contraintes de court et long chemin définies pour l'algorithme *RCV*. Nous proposons une nouvelle contrainte B_s associée à chaque groupe s, qui définit la différence maximale de délai acceptable (marge) après routage entre le plus grand délai et le plus petit délai des *net* de s. Cette marge est la contrainte qui permet de définir l'équilibrage au sein d'un groupe. C'est à partir de la marge disponible pour chaque groupe que les contraintes de délai minimum et maximum seront extraites par l'algorithme (en accord si besoin avec celles définies par l'utilisateur), à partir des délais minimum possible par la technologie utilisée. Ce délai minimum est obtenu à l'étape 2 de l'algorithme 4.1. Soit d_n le délai du *net* n après routage à l'étape 1, et d_{tn} le délai cible associé, tel que calculé par la fonction de coût utilisée par *RCV* (figure 4.2.1).

Étant donné un groupe s comprenant N_s *net*, chaque *net* $n \in s$ possède un chemin $P_n = [e_1^n, \ldots e_i^n, \ldots e_{k_n}^n]$ avec k_n arcs utilisés (e_i^n est le i^e arc utilisé par le *net* n), et ses contraintes c_s définissant $d_{\min_n}$, $d_{\max_n}$ et la marge B_s. Tous les *net*

de s sont dits équilibrés si et seulement si les deux conditions suivantes sont respectées :

1. le délai de chaque chemin de s est compris dans l'intervalle défini par B_s, c'est à diremaxD(s) − $B_s \leq d_n \leq$ maxD(s) pour chaque $n \in N_s$**,** avec maxD(s) le plus grand délai de l'ensemble s.
2. le délai de chaque chemin de s est à l'intérieur des contraintes c_n, c'est-à-dire $d_{\min_n} \leq d_n \leq d_{\max_n}$ pour chaque n tel que $n \in N_s$.

Le problème de routage de ce chapitre est formulé comme suit. Étant donné N_s *net* appartenant à s, calculer un chemin $P_n = [e_1^n, \ldots e_k^n]$ pour chaque *net* $n \in s$ de telle sorte que le délai de tous les chemins soient équilibrés tel que défini par les 2 précédentes conditions, et respecter les objectifs suivants dans l'ordre :

1. ne pas utiliser le même arc pour deux *net* différents (absence de conflits), c'est-à-dire $\forall (k_1,k_2),(n_1,n_2)$, $\{n_1 \neq n_2, \nexists (e_{k_1}^{n_1}, e_{k_2}^{n_2}) : e_{k_1}^{n_1} = e_{k_2}^{n_2}\}$;
2. minimiser la somme totale des délais de chaque chemin du *netlist* $\sum_{n=1}^{N} d_n$;
3. minimiser la consommation des ressources de routage (somme totale des longueurs des arcs $\left|\lambda_{e_i^n}\right|$) $\sum_{n=1}^{N}\sum_{i=1}^{k}\left|\lambda_{e_i^n}\right|$

Cet ordre correspond aux contraintes de routage utilisés par l'algorithme *RCV*. Celui-ci considère les contraintes associées à chaque *net* de façon indépendante. Ainsi, chaque *net* est routé de façon à remplir les contraintes de petit et grand délais, mais l'équilibrage au sein d'un groupe n'est en général pas réalisé. *RCV* ne vise pas à réaliser un équilibrage, et il doit être étendu pour réaliser de façon robuste un équilibrage. Cependant, *RCV* réalise l'équilibrage de groupes de taille variables lorsque deux conditions précisent sont réunies :

1. l'union des contraintes c_n pour un groupe s respectent la contrainte B_s (ce qui est toujours vrai lorsque toutes les contraintes du groupe sont identiques) ;
2. l'ensemble des *net* du groupe s sont routés sans conflit, dans les contraintes spécifiées.

Ces deux conditions sont cependant très restrictives, et rendent l'équilibrage de délai peu robuste. Cette faiblesse s'explique par la figure 4.2.2 qui présente une situation classique de l'équilibrage de délais. Le chemin le plus à droite sur la figure, qui possède le délai $\Delta_{\max}$ le plus grand du groupe, est également le plus contraint. En effet, ce *net* possède physiquement le délai le plus proche de $d_{\max}$. Pour peu que l'utilisateur ait spécifié une contrainte d'équilibrage B_s égale ou très proche de $\Delta_{\max} - \Delta_{\min}$, les possibilités de routage de ce *net* sont très restreintes. Dans une zone congestionnée, il n'est pas rare qu'aucune solution non-conflictuelle ne soit possible pour plusieurs *net* en conflit mutuels, possédant des contraintes similaires. Or, la contrainte d'équilibrage, d'un bus de communication par exemple, permet aux circuits électronique un échantillonnage adéquat dans le temps des signaux. Si cette contrainte n'est pas respectée, les données sont erronnées. Déclarer l'échec de routage dans une telle situation oblige l'utilisateur à modifier la position des composants. De plus, la problématique n'est pas forcément simple lorsque la surface du *WaferIC* est densément occupée. La problématique est similaire à celle sur *FPGA* : l'utilisateur spécifie en général une fréquence de fonctionnement objectif, qui n'est pas nécessairement réalisable par les outils. Plutôt que d'abandonner, les outils commerciaux tentent

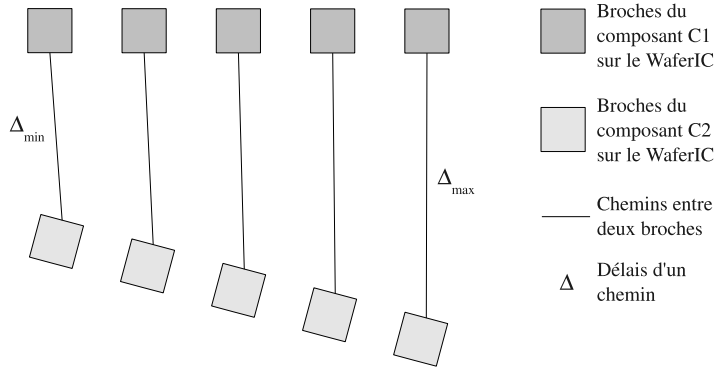


FIGURE 4.2.2: Situation générale de l'équilibrage d'un bus : deux composants (C1 et C2) se font face, mais les délais optimaux de chaque *net* ne sont pas égaux.

en général de terminer les étapes de placement et routage, quitte à réduire quelque peu la fréquence finale du circuit. Cela permet deux choses : d'abord, de proposer un circuit probablement fonctionnel, bien que hors des contraintes spécifiées. Pour une plateforme de prototypage, il est justifé de démontrer une application fonctionnelle, même limitée, qu'un circuit non fonctionnel. Surtout, finaliser le traitement donne plus d'informations sur les difficultées rencontrées par l'outil : l'utilisateur possède donc plus d'informations pour améliorer le circuit et ainsi atteindre les objectifs visés.

De même, le prototypage sur *WaferIC* visera un certain nombre de critères, notamment de fréquence de fonctionnement. Ces différents objectifs se traduisent notamment par les contraintes de délai minimum et maximum, et d'équilibrage. Cependant, l'algorithme de routage étant séquentiel, il doit prioriser une des deux contraintes, c_n ou B_s. Dans l'exemple très courant de la figure 4.2.2, prioriser c_n revient donc à abandonner le routage lorsqu'aucune solution non conflictuelle qui respecte c_n n'est trouvée pour le chemin ayant le plus grand délai. En effet, si la contrainte d'équilibrage est spécifiée, elle est plus importante que la contrainte c_n. Sinon, pourquoi l'utilisateur spécifie-t-il B_s, tandis que c_n permet d'obtenir un premier équilibrage lorsque tous les c_n du groupe sont égaux ? De la même manière que les routeurs pour *FPGA* relaxent la contrainte de fréquence pour proposer un circuit fonctionnel plus lent, nous proposons que le routage pour le *WaferIC* s'autorise, lorsque nécessaire, à relaxer la contrainte c_n pour respecter B_s, lorsque la contrainte B_s est spécifiée. Il est important de comprendre que la contrainte c_n doit être conservée, elle sert de guide à l'algorithme A*.

Pour réaliser l'équilibrage de groupes de *net*, une solution naïve consiste à appliquer l'algorithme 4.2, entre les étapes 4 et 5 de l'algorithme 4.1. Cet algorithme naïf prend une liste de *net* à équilibrer N (un *netlist*) associée à des contraintes C. Pour chaque groupe s_i de N, il trouve le chemin ayant le plus grand délai, noté $\mathrm{maxD}(s_i)$. Par construction de l'algorithme 4.2, à la première itération chaque chemin possède le délai minimal possible pour la technologie ; le chemin ayant le plus grand délai dans s_i ne peut être raccouci. Ensuite, la contrainte de délai minimum est mise à jour pour chaque élément de la liste, qui répond à l'union des contraintes d'équilibre B_{s_i} et de délai minimum $d_{\mathrm{min_i}}$ originelle, qui est donc $\max(d_{\mathrm{min_i}}, \mathrm{maxD}(s) - B_{s_i})$. La contrainte de délai maximum est également mise à jour en fonction de B_{s_i}

Algorithm 4.2 Un algorithme simple de préparation des contraintes de délai pour l'équilibrage d'un groupe de taille arbitraire des délais.

```
1: procedure CONSTSETUP(N, C)
2:     for all groupe s_i du netlist N qui n'est pas entièrement équilibré do
3:         M ← maxD(s_i)                                  ▷ Trouve le délai maximum de s_i
4:         for all net dans s_i do
5:             d_min_n ← max(d_min_n, max(0, M − B_s))
6:             d_max_n ← M
7:         end for
8:     end for
9: end procedure
```

Algorithm 4.3 Algorithme de relaxation de constraintes.

```
1: procedure CONSTSETUP(N, C, I, maxI)
2:     for all groupe s_i du netlist N qui n'est pas entièrement équilibré do
3:         for all net n dans s_i do
4:             u ← B_{s_i} × I                            ▷ Incrément à ajouter à d_min_n
5:             if n n'est pas le plus long du groupe then
6:                 if d_n > d_min_n then d_min_n ← d_n
7:                 else d_min_n ← d_max_n − u
8:                 end if
9:             end if
10:            if d_max_n ≤ d_n then
11:                d_max_n ← d_n + u
12:            end if
13:        end for
14:        for all net n in s do          ▷ Met à jour min/max pour les autres membres du groupe si besoin
15:            if d_min_n < maxD(s) − B_s then
16:                d_min_n ← maxD(s) − B_s
17:            end if
18:            if d_max_n − d_min_n < B_s then
19:                d_max_n ← d_min_n + B_s
20:            end if
21:        end for
22:    end for
23: end procedure
```

et du plus grand délai du groupe $\Delta_{\max_i}$. L'algorithme peut ainsi réduire le délai $d_{\max_i}$ pour les *net* dont le délai est en conflit avec B_{s_i}. En effet réduire $d_{\max_i}$ ne rentre jamais en conflit avec la contrainte spécifiée par l'utilisateur à l'origine. Or, cet algorithme ne trouve pas de solution dès qu'un chemin ne peut être routé dans les contraintes ainsi calculées. Comme expliqué précédemment, il peut arriver qu'aucune solution ne soit possible pour le chemin de plus grand délai dans les zones congestionnées, et ce quelque soit l'algorithme utilisé. Aucune *netlist* utilisée pour l'équilibrage des délais n'était routée avec succès total lorsque la densité moyenne atteint 10 %. La technique du *ripup and reroute*, » telle que présentée à la section 2.1, est également limitée : en plus de temps de routage conséquent, il est également possible que deux *net* avec de telles contraintes entrent en conflit, qui ne peut être résolu autrement qu'en relaxant la contrainte de délai maximum pour un des *net* en conflit. Cependant, cela implique de mettre à jour les contraintes de délai minimum de tous les éléments du groupe s_i.

L'algorithme 4.3 propose une solution à un tel problème. Il relaxe itérativement les contraintes de délai minimum et maximum, et ce pour chaque membre d'un groupe. Comme d'autres routeurs séquentiels, une approche «*ripup and reroute* » est utilisée pour résoudre les conflits (étape 5 de l'algorithme 4.1) et itère jusqu'à ce qu'une condition d'arrêt soit activée. Par exemple, l'arrêt peut se produire lorsque tous les *net* sont routés, ou qu'un nombre maximal d'itérations maxI est atteint. L'algorithme 4.3 est intégré au sein de ces itérations, et relaxe les contraintes d'équilibrage par étapes sur chaque ensemble non-équilibré. L'incrément ajouté au délai (défini ligne 4) a une forme linéaire : d'autres fonctions pourraient être utilisées. La fonction linéaire a donné de bons résultats, comme montrés à la section 4.3. L'optimalité de cette fonction d'incrément reste une question ouverte.

La boucle principale, qui commence à la ligne 3 de l'algorithme 4.3, met à jour en priorité la borne inférieure (délai minimum) de chaque *net* de l'ensemble concerné (ligne 6). Ce choix est lié à la fonction de coût de *RCV*, qui cible les routes qui sont très proches de la borne inférieure. En effet la figure 4.2.1 est très schématique, et ne reflète pas parfaitement la réalité. Le point de coût minimum (d_{target}) est en réalité très proche du délai minimum accepté (d_{min}). Le A* a donc tendance à rester bloqué sur une zone proche du délai minimum sans trouver de solution, ce qui ralentit fortement le temps de routage. Fung *et al* ont rapporté le phénomène dans [38], qui permet également de limiter le nombre de fils utilisés dans un *FPGA*. Cependant, *RCV* route chaque *net* indépendamment les uns des autres. Le nombre d'itérations de chaque *net* est donc plus faible que pour réaliser un équilibrage : en effet, l'équilibrage d'un groupe peut nécessiter de router plusieurs fois chaque membre jusqu'à trouver une solution équilibrée. Ceci est d'autant plus difficile que la zone est congestionnée, et le groupe de taille importante et les contraintes temporelles sont sévères. Cette problématique n'est pas rencontrée par *RCV*. L'approche proposée priorise le temps de routage. La contrainte de délai minimum est augmentée en priorité afin de forcer l'algorithme A* à rechercher des trajets plus longs. Ensuite (ligne 10), si le chemin pour le *net n* possède un délai plus important que la contrainte de délai maximale, alors d_{max_n} est relaxé d'un incrément u.

La dernière boucle (ligne 14) met à jour les contraintes des autres membres du groupe, de telle sorte que la contrainte B_s soit satisfaite. Pour cela, l'algorithme augmente la borne de délai minimum des membres qui sont maintenant en dehors de la limite définie par B_s. Par exemple, supposons que le délai minimum d_{min_n1} du *net* n_1 a été mis à jour par la première boucle. Un autre *net* n_2 du même ensemble peut maintenant avoir un délai minimum d_{min_n2} qui est en dehors de la contrainte B_s, c'est-à-dire $d_{\text{min}_\text{n2}} < d_{\text{min}_\text{n1}} - B_s$. Il est nécessaire de défaire le routage de tous les membres qui sont situés en dehors de la nouvelle frontière, pour les router de nouveau avec les nouvelles contraintes lors de la prochaine itération. La fonction optimale d'augmentation du délai est un problème ouvert, et la fonction linéaire (par incréement à chaque itération) a été choisie pour sa simplicité.

Une solution d'allocation dynamique des contraintes de délai a été décrite dans cette section. Elle se base sur l'algorithme *RCV* publié en 2008 [38], et y ajoute les composants nécessaires pour l'équilibrage de groupes. L'algorithme A* est utilisé pour trouver des solutions non-conflictuelles équilibrées, grâce à la fonction de coût de *RCV*. Cependant,

l'équilibrage augmente considérablement les temps de calculs, comme montré au tableau 4.2. Cette observation est corrélée avec ce qui est rapporté par les documents techniques des fabricants de *FPGA*, dont les outils supportent cette fonctionnalité [107, 108]. L'algorithme A* constitue le goulot d'étranglement : un algorithme complémentaire ainsi qu'une modification de sa fonction de coût sont proposés à la section suivante, pour réduire significativement le temps de routage. En effet, la technologie ciblée a des contraintes serrée en ce qui concerne les temps de calcul (moins de 10 minutes pour un *netlist* dense), mais possède également une structure régulière qui est mise à profit pour accélérer les calculs.

4.2.2 Algorithme d'équilibrage des délais par table d'équivalences

Le problème de routage défini à la section 4 peut être considéré comme un problème d'ajustement des délais pour un ensemble de *net*. Après le calcul d'une solution optimale en termes de délais pour chaque *net* (étape 2 de l'algorithme 4.1), les étapes de gestion des conflits, et de l'équilibrage de délai a lieu. Pour cette dernière étape, fortement itérative, l'algorithme proposé dans cette section est inspirée de techniques pour *PCB* dites "qui serpentent". L'algorithme est appliqué au réseau d'interconnexions du *WaferBoard*, et utilise une tableau d'équivalences pour une résolution rapide. Il est exécuté à chaque itération avant le lancement de l'algorithme A* (voir section 2.2.2) pour réduire l'ensemble des *net* non-équilibrés à résoudre. Dans ce qui suit, le modèle de délai utilisé est décrit, puis l'utilisation et la construction de la table d'équivalences sont expliqués.

L'approche proposée suppose qu'il existe une relation linéaire entre le délai associé à un arc du graphe (cf. section 1.1) et sa longueur. C'est en effet le cas pour les réseaux d'interconnexions sur silicium couplés à l'utilisation de répéteurs espacés régulièrement. Le délai total d'un chemin qui utilise k arêtes dépend :

1. du nombre de sommets $k+1$;
2. de d_c, la constante de délai qui existe lors du franchissement d'un sommet (*crossbar*) ;
3. du délai d'un arc d_w pour un arc de longueur 1.

Le paramètre d_w définit la relation proportionnelle directe entre le délai d'un arc et sa longueur, soit la distance entre les deux sommets raccordés. Dans le cadre de cette application, d_c est environ dix fois plus grand que d_w. En conséquence, le retard d_u entre deux sommets reliés par une arête est donnée par l'équation 4.2.1 où λ est la distance cumulée parcourue par le chemin, entre les paires de sommets source v_s et destination v_d,

$$d_u = \lambda d_w + d_c \tag{4.2.1}$$

Le délai total d'un chemin p en utilisant k arcs est développé dans l'équation 4.2.3 :

$$d_p = d_c + \sum_{i=1}^{k} d_{u_i} = d_c + \sum_{i=1}^{k} (\lambda_i d_w + d_c) \tag{4.2.2}$$

$$d_p = (k+1)d_c + d_w \sum_{i=1}^{k} \lambda_i = (k+1)d_c + d_w D_n \tag{4.2.3}$$

Le délai du chemin p est proportionnel au nombre d'arcs utilisés et de leur longueurs respective λ_i. Par conséquent, il est possible d'ajuster le délai d'un chemin en modifiant le nombre de sommets et la longueur des arcs utilisés. Il est donc possible de diviser un ou plusieurs arcs par d'autres arcs plus courts, d'effectuer le calcul à l'avance et de le réutiliser d'un *net* à l'autre. Ce calcul rapide évite de ré-appliquer l'algorithme A* (très générique mais aussi bien plus lent) pour l'ensemble des *net* routés avec succès par cet algorithme. Par exemple, supposons un chemin p avec un délai total d_{p_1} qui utilise $k_1 = 2$ segments de longueur 2 et 4. Pour augmenter le délai, le dernier segment (longueur 4) peut être divisé en deux segments identiques de longueur 2. De l'équation 4.2.3 et des définitions précédentes, le nouveau chemin construit p_2 utilise maintenant $k_2 = 3$ arcs, et le délai total d_{p_2} est de :

$$d_{p_2} - d_{p_1} = d_c(k_2 + 1 - k_1 - 1) + d_w \left(\sum_{i=1}^{k_2} \lambda_{2i} - \sum_{i=1}^{k_1} \lambda_{1i} \right) \tag{4.2.4}$$

$$\Rightarrow d_{p_2} - d_{p_1} = d_c(3 + 1 - 2 - 1) + d_w(6 - 6) = d_c \tag{4.2.5}$$

L'équation 4.2.4 exprime l'augmentation du délai du chemin lorsque l'on remplace un arc par d'autres, dans un réseau d'interconnexions sur silicium. Il est possible de calculer un grand nombre de possibilités rapidement, et une version partielle des possibilitées est montrée au tableau 4.1. Ces résultats montrent le délai qui est ajouté à un chemin par la subdivision de chaque longueur λ (colonne 1) en différents arcs. Par exemple, un arc de longueur 2 (première colonne, ligne 2) peut être divisé en 2 segments de longueur 1, pour un allongement de $1d_c$ (colonne 2, ligne 3). De même, un segment de longueur 16 (colonne 1, ligne 6) peut être divisé en 4 segments de longueur 4, ajoutant un délai de $3d_c$. Il est bien sur possible de subdiviser en combinaisons plus complexes comme par exemple un arc de longueur 16 en arcs de longueurs 8, 4, 1, 1, 1, 1. L'exemple donné au tableau 4.1 est simpliste comparativement à ce qui est généré et utilisé par l'algorithme. Subdiviser chaque longueur par un ensemble d'arcs plus courts permet de construire la moitié inférieure du table 4.1.

Il est en effet possible de réaliser l'opération inverse : à partir d'un arc de longueur 2, construire un chemin avec les arcs de longueur 4 et 2. Il convient d'utiliser des arcs de directions opposées, sous peine de ne plus relier les même sommets source-destination. Cependant, de telles constructions permettent d'obtenir des augmentations de délai plus variées, et offrent donc une panoplie de diverses solutions.

La réduction de l'espace de recherche par rapport au A*peut résulter en des solutions de moindre qualité, mais l'utilisation d'une table est bénéfique pour la réduction du temps de calcul, résultats présentés à la section 4.3.

Table 4.1: Tableau partiel de l'augmentation du délai (normalisé), via le «découpage» d'un segment de longueur λ (première colonne) en plusieurs autres (colonnes 2-7). Les valeurs sont normalisées par rapport à l'augmentation minimale du délai d_c accessible à la technologie du WaferBoard.

λ	1	2	4	8	16	32
1	x	$1.26d_c$	$1.66d_c$	$2.46d_c$	$4.04d_c$	$7.22d_c$
2	$1d_c$	x	$1.53d_c$	$2.32d_c$	$3.91d_c$	$7.09d_c$
4	$3d_c$	$1d_c$	x	$2.06d_c$	$3.65d_c$	$6.93d_c$
8	$7d_c$	$2d_c$	$1d_c$	x	$3.12d_c$	$6.3d_c$
16	$15d_c$	$7d_c$	$3d_c$	$1d_c$	x	$5.24d_c$
32	$31d_c$	$15d_c$	$7d_c$	$3d_c$	$1d_c$	x

Algorithm 4.4 Algorithme rapide d'équilibrage par table d'équivalences.

```
1:  procedure FASTBALANCE(N, C, LUT)
2:      for all net dans s do
3:          for all solution p dans LUT avec d_p ≥ d_min_n et d_p ≤ d_max_n do
4:              if p est applicable then                    ▷ arc existe dans le chemin courant ?
5:                  c ← 0
6:                  while p est en conflit et c < maxPerm do
7:                      p ← nextPerm(p)                     ▷ Calcul par permutations
8:                      c ← c + 1
9:                  end while
10:             end if
11:             if s est valide then Enregistre p
12:             else Router p avec A*                       ▷ utilise la fonction coût de RCV
13:             end if
14:         end for
15:     end for
16: end procedure
```

L'algorithme effectue la construction de la table d'équivalences avant routage, une seule fois. La construction de la table se fait par récursivité : par subdivision de la plus grande longueur du réseau d'abord (moitié inférieure du table 4.1), puis par allongement récursif (moitié supérieure). La table complète pour le réseau ciblé comprend plus de 500 entrées, chaque entrée étant triée selon la valeur du délai apporté.

L'algorithme 4.4 correspond à une sous partie de l'étape 5 de l'algorithme 4.1, c'est à dire le routage des *net* ayant des contraintes d'équilibrage. Pour chaque groupe *s* à équilibrer (boucle principale, ligne 2), la table d'équivalences est solicitée. Les solutions potentielles sont récupérées (ligne 3), filtrées selon les bornes de délai associée au *net* en cours de routage. L'algorithme vérifie que la solution est bien applicable : par exemple, segmenter un arc de longueur 2 en deux arcs de longueur 1 exige la présence d'un arc de longueur 2 dans la solution initiale. Le test de la ligne 4 se charge de cette vérification. Un des intérêts majeurs du calcul par table d'équivalences, est la possibilité d'utiliser le routage par permutations présenté au chapitre 2.2.6 pour chaque solution potentielle de la table d'équivalence. Les lignes 5 à 9 font donc appel aux routines de calcul de permutations, ce qui démultiplie les possibilités offertes par la table de solutions.

Si une solution est trouvée rapidement, celle-ci est enregistrée. Sinon, un appel est fait pour réaliser le routage par l'algorithme A*, en utilisant la fonction de coût de *RCV*.

4.2.3 Version tronquée de la fonction coût de RCV

Les observations réalisées lors du routage avec équilibrage ont montré qu'énormément de temps était passé dans l'algorithme A*. L'approche basée sur une table d'équivalences réduit significativement le nombre de *net* calculés par celui-ci. Un nombre important nécessitent cependant une exploration plus poussée dans l'espace des solutions. Effectivement, le A* réalise une recherche très exhaustive, ce qui explique sa lenteur par rapport au routage basé sur un patron, tel que le routage par permutations. Dans cette section, une modification de la fonction de coût de *RCV* est proposée, pour restreindre l'espace de recherche. Cette modification n'entraîne pas de conséquences quant à la qualité du routage final (délai et longueur totale des chemins).

Par construction algorithmique, aucune route qui nécessite un équilibrage des délais ne peut dépasser la contrainte de délai maximum. Si une route pour un *net n* ne respecte pas les contraintes de délai maximum, il est très probable qu'un nombre important de *net* qui font partie de l'ensemble *s* devront être enlevés, puis re-routés. En effet, les *net* sont routés proches de leur contrainte de délai minimum (autant que possible pour limiter la consommation de ressources). Un dépassement de la contrainte de délai maximum par un seul *net* implique donc le routage d'un grand nombre de *net* du groupe *s*. Dans ce cas, les bornes de délai seront relaxées, et de nouvelles solutions calculées. Ce processus est répété jusqu'à ce que tous les *net* de l'ensemble vérifie l'équilibrage, et qu'aucun ne soit en conflit avec d'autres *net*. Ainsi, il est inutile pour le A* d'explorer les solutions au delà de la zone de délai maximum.

Empêcher l'algorithme A* d'explorer les solutions qui dépasseront nécessairement les contraintes de délai est réalisé comme suit. Le A* utilise une heuristique de prédiction du coût (délai) lors du choix d'un nœud, au cours de la construction d'un chemin. Cette heuristique est dite complète si et seulement si elle garantit que, pour un nœud donné, le délai du chemin utilisant ce nœud ne peut jamais être inférieur à celui prédit. Le délai prédit est appelé ici $d_{\text{prédit}}$. De plus, le A* garde en mémoire le délai de la source jusqu'au nœud courant d_{courant} (ce délai est mesuré, et non prédit). Notre implémentation utilise une heuristique admissible (cf. section 2.2.2), ainsi le A* ne doit pas explorer un nœud dont la somme des délais $d_{\text{prédit}}$ et d_{courant} dépasse la contrainte de délai maximal. Si cette condition est vraie pour le nœud exploré, alors le coût de celui-ci est infini, détournant le A* d'une exploration coûteuse. La figure 4.2.3 illustre la fonction de coût proposée pour l'équilibrage des délais, ainsi que celle utilisée par *RCV*.

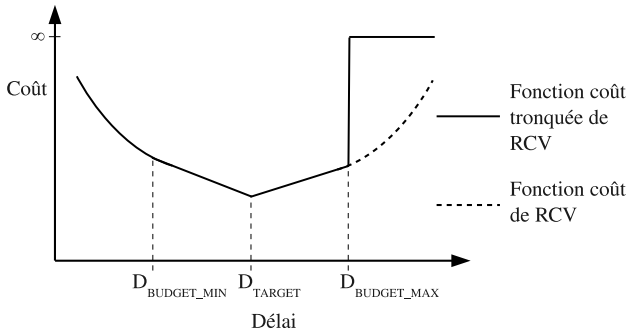


FIGURE 4.2.3: La fonction de coût proposée : une troncature de la fonction utilisée par *RCV*. Ainsi, l'espace de recherche est réduit. La fonction utilisée par *RCV* est en pointillés, pour faciliter la comparaison.

Une telle approche ne peut être utilisée pour les nœuds n'ayant pas besoin d'être équilibrés. En effet, cette fonction pousse le A* à abandonner un vaste espace de recherche, qui est nécessaire dans les zones congestionnées. Il n'est ainsi pas possible d'utiliser une telle fonction efficacement pour des *net* ayant des contraintes de délai maximum, mais sans équilibrage.

L'impact de cette approche sur le temps de routage est présenté à la section suivante.

4.3 Résultats des approches proposées

Cette section présente les résultats des approches proposées, les accélérations et la qualité de routage de l'approche par table d'équivalence, l'utilisation des permutations au sein de l'équilibrage, et l'impact de la fonction de coût proposée.

Il n'existe pas, aux meilleures de nos connaissances, de banc d'essai pour le placement et routage pour *PCB*. Cet état de fait a d'ailleurs été souligné par Abboud *et al* [85], et très peu de circuits pour *PCB* sont publiquement accessibles. De plus, il n'y a aucun algorithme d'équilibrage de délais publié qui utilise un réseau d'interconnexions sur silicium, encore moins pour un réseau similaire au *WaferBoard* : les comparaisons auprès d'autres outils sont sinon impossibles, au moins impraticables. Les expériences présentées dans cette section ont été réalisées à l'aide de bancs d'essais générés à partir du modèle présenté au chapitre 4.4. Les contraintes d'équilibrage ont été appliquées à tous les bus de chaque *netlist*, selon deux schémas. Le premier avec un équilibrage fixé pour tous, le deuxième choisit aléatoirement entre l'équilibrage minimal possible par la technologie, et 25 fois ce minimum. Trois catégories de bancs d'essais ont été ensuite construits. La première catégorie est appelée « minimum», et correspond à router l'ensemble des bus avec un équilibrage égal au minimum de la technologie. La deuxième est appelée « raisonnable », et correspond à une charge de travail jugée probable pour l'utilisation de la technologie. L'équilibrage est fixé à 3 fois l'équilibrage minimal réalisable. Enfin, la troisième catégorie appelée « relaxée » correspond à des contraintes aléatoires appliquée sur chaque bus, comprises entre 1 et 25 fois le minimum. La largeur des bus est comprise entre 2 et 30 *net*, et représente

environ 25 % des *net* à router pour chaque *netlist*. Le taux d'occupation est exprimé en pourcentages, et correspond au nombre *moyen* de broches par nœud du graphe, sur l'ensemble de la surface de routage. La machine utilisée pour extraire les résultats utilise un processeur intel i7 à 3Ghz, avec un système d'exploitation GNU/Linux 64 bits.

Les temps de calcul et les violations de contraintes sont montrés au tableau 4.2. Ces résultats sont extraits sans utiliser la fonction coût proposée, ceux-ci étant présentés par la suite. Le temps de routage est comparé à l'utilisation de l'algorithme A* avec et sans l'équilibrage. La colonne S-LUT présente les résultats lorsque la moitié inférieure du table 4.1 est utilisée. La colonne C-LUT fait au contraire un usage complet de la table. L'algorithme basé sur la table d'équivalences, qu'elle soit complète ou simplifiée, accélère le routage par un facteur compris entre 1.54 (1.71 pour la table complète) et 4.21 (4.24). Ce résultat est significatif, mais ne permet pas d'obtenir les temps de routages désirés. En effet, si l'on se cantonne à regarder les résultats pour les *netlist* de densité attendue (maximum ~30%) et un équilibrage raisonable, le temps de calcul est d'environ 45 minutes, soit bien au delà des requis de l'application, qui visent moins de 10 minutes. Les *netlist* plus denses, ou avec des contraintes plus fortes, peuvent prendre jusqu'à 2 h 45 pour un routage complet.

Un aspect très positif des résultats est le nombre de violations : celui-ci est toujours à 0, et le routage se termine toujours avec succès sur les *netlist* testées. L'algorithme compte les violations d'équilibrage de délai, et non pas de délai maximum, tel qu'expliqué à la section 4.2.1. Ainsi, l'algorithme a trouvé une solution équilibrée pour chaque *netlist* testée, mais au prix d'un délai plus important. Ce comportement désiré suppose que les contraintes d'équilibrage sont prioritaires à celles de délai maximum.

En ce qui concerne les permutations, la figure 4.3.1 présente l'accélération apportée par l'utilisation de cette approche, pour les même *netlist* et densités présentés au tableau 4.2. Couplée au routage par table d'équivalences, les permutations permettent de gagner entre 4 % et 33 % du temps de calcul, sauf dans le cas d'un *netlist* très dense (au delà de ce qui est attendu, limité à ~30 %). Les résultats de la table 4.1 tiennent compte de l'accélération apportée par la technique des permutations. Cependant, même dans le cadre de *netlist* au delà des densités attendues, l'accélération est visible.

La somme des délai totaux des routes est une métrique couramment utilisée pour mesurer la qualité des solutions générées par les routeurs. Elle est présentée au tableau 4.3. L'influence de l'utilisation de la table d'équivalences sur la qualité du routage, comparée au A* utilisé seul, est montrée. La colonne A* réfère à un routage sans table d'équivalences, et la somme du délai total de toutes les routes est en picosecondes. La somme des délais totaux lorsque le table d'équivalences est utilisée (simple ou complète) est rapportée en pourcentage d'augmentation par rapport au A*seul. Tel qu'attendu, la table d'équivalences influence la qualité générale du routage : cependant, l'impact est compris entre 2.76 % et 7.43 % au maximum pour la table simple, et 2.76 % à 7.30 % lorsque l'on utilise la table complète. Devant l'accélération proposée par ces solutions, le gain en vaut probablement toujours la perte de qualité dans le cadre de l'application de prototypage qu'est le WaferBoard. Cependant, l'utilisation reste optionnelle en fonction des besoins

de l'utilisateur.

L'impact de la modification de la fonction de coût est également présenté dans ces résultats. Le temps de routage total pour les différents *netlist* et contraintes d'équilibrage est montré à la figure 4.3.2. Le facteur d'accélération comparé à la meilleure solution précédente (routage avec C-LUT) est également visible. Le profil d'accélération va décroissant avec la densité : les itérations très nombreuses exigées par les *netlist* de 40 et 50 % (qui se situent au delà des densités attendues), ne permettent pas à l'approche d'accélérer les calculs. Par contre, les *netlist* de densité 30 % sont accélérées d'un facteur 10 (équilibrage minimal et raisonable) à 5.5 (équilibrage relaxé). Les *netlist* les moins denses sont très fortement accélérées, avec une accélération maximale obtenue de 400 fois.

Du côté des temps de calcul, l'équilibrage des délais est effectivement très rapide pour les *netlist* faiblement denses. L'algorithme proposé route des *netlist* réalistes (10 % de densité) avec équilibrage en moins de 10 secondes, voir 3 secondes si l'équilibrage est raisonable. Les *netlists* très denses (de 30 %) sont routées en à peine plus de 8 minutes lorsque l'équilibrage est raisonnable, et 15 minutes environ lorsque les contraintes sont très fortes. Les autres bancs d'essai voient les temps augmenter, mais restent à la portée d'une machine actuelle, pour peu que l'utilisateur soit patient.

Cette section a montré que l'équilibrage de délais énoncé ne peut être réalisée dans les temps impartis avec un algorithme A*simple. Cependant, les techniques proposées (routage par table d'équivalence, permutations et fonction de coût) permettent mises ensemble de dramatiquement réduire les temps de routage. Si l'on prend l'exemple d'une *netlist* de densité importante (20 %) avec un équilibrage raisonnable, un A*seul mettra sur le *WaferIC* 3479 s (soit environ 58 minutes, à comparer aux 10 minutes visées). Utiliser la table d'équivalences réduit le temps de calcul à 1890 s (soit donc 32 minutes), autrement dit une accélération de 1.84. Cette approche réduit légèrement la qualité de routage, mais permet l'utilisation de permutations en plus, faisant descendre le temps de calcul à 1421 s (soit 23 minutes). Finalement, la dernière approche proposée, la modification de la fonction de coût sans perte de qualité, fait descendre le temps de calcul à 14 s (0.23 minutes), un facteur d'accélération de 100 (!). Il est possible pour l'utilisateur qui désire un routage de la meilleure qualité possible de ne pas utiliser la table d'équivalences (ce qui supprime les permutations). Dans ce cas, le temps de calcul sera d'environ 1 minute, encore largement sous les 10 minutes visés.

Cet exemple permet de comprendre l'intérêt de chaque proposition, et la pertinence de ces recherches devant les temps excessifs d'un algorithme A*lorsque utilisé seul.

Tableau 4.2: Résultats de routage avec équilibrage pour les approches basées sur la table d'équivalences.

		Temps de routage (secondes) (facteur d'acc. face au A* utilisé seul)									
		A* sans éq.	1-25 ("relaxé")			1 ("minimal")			3 ("raisonnable")		
		-	A*	S-LUT	C-LUT	A*	S-LUT	C-LUT	A*	S-LUT	C-LUT
Densité de *netlist*	10	0.4	421	199 (2.12)	192 (2.19)	11153	3977 (2.80)	3670 (3.03)	1565	1013 (1.54)	913 (1.71)
	20	3.75	854	411 (2.08)	397 (2.15)	24117	8200 (2.94)	7523 (3.20)	3479	2176 (1.60)	1890 (1.84)
	30	9.1	1208	643 (1.88)	618 (1.95)	37456	13317 (2.81)	12727 (2.94)	6177	3769 (1.64)	3441 (1.80)
	40	61	3032	1437 (2.11)	1408 (2.15)	53979	18677 (2.89)	18070 (2.99)	14195	6169 (2.30)	5619 (2.53)
	50	283	5634	2654 (2.12)	2771 (2.03)	88921	29019 (3.06)	28925-(3.01)	39238	9314 (4.21)	9247 (4.24)

Tableau 4.3: Influence des différentes approches basées sur la table d'équivalences sur le délai total des routages générés.

		Délai total (ps) pour le A* et augmentation relative du délai avec équilibrage par table d'équivalences (%)								
		1-25 ("relaxé")			1("minimal")			3 ("raisonnable")		
	Éq.	A*	S-LUT	C-LUT	A*	S-LUT	C-LUT	A*	S-LUT	C-LUT
Densité de *netlist*	10	348 10^6	3.04	2.87	351 10^6	7.05	6.86	349 10^6	6.62	6.24
	20	703 10^6	2.80	2.68	699 10^6	7.43	7.30	699 10^6	6.32	5.98
	30	1048 10^6	3.00	2.93	1046 10^6	7.17	7.12	1048 10^6	6.33	6.12
	40	1404 10^6	2.81	2.76	1047 10^6	6.56	6.57	1400 10^6	6.08	6.01
	50	1769 10^6	2.76	2.76	1758 10^6	5.88	5.86	1767 10^6	5.33	5.31

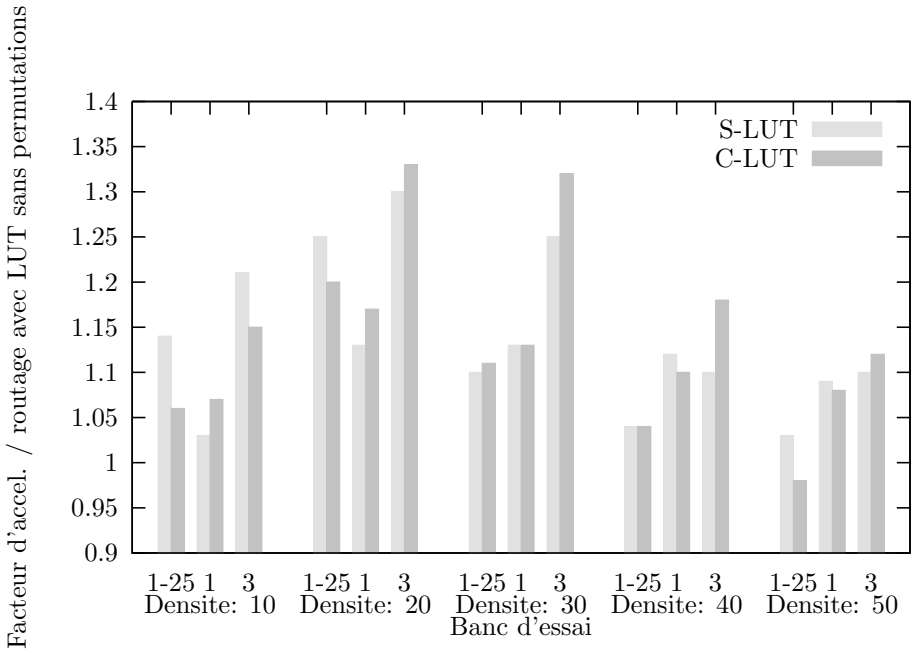


FIGURE 4.3.1: Facteur d'accélération sur le temps de calcul apporté par le calcul des permutations, appliqué avec les deux approches de routage par table d'équivalences. Les permutations apportent pratiquement toujours une accélération, parfois jusqu'à un facteur 1.33.

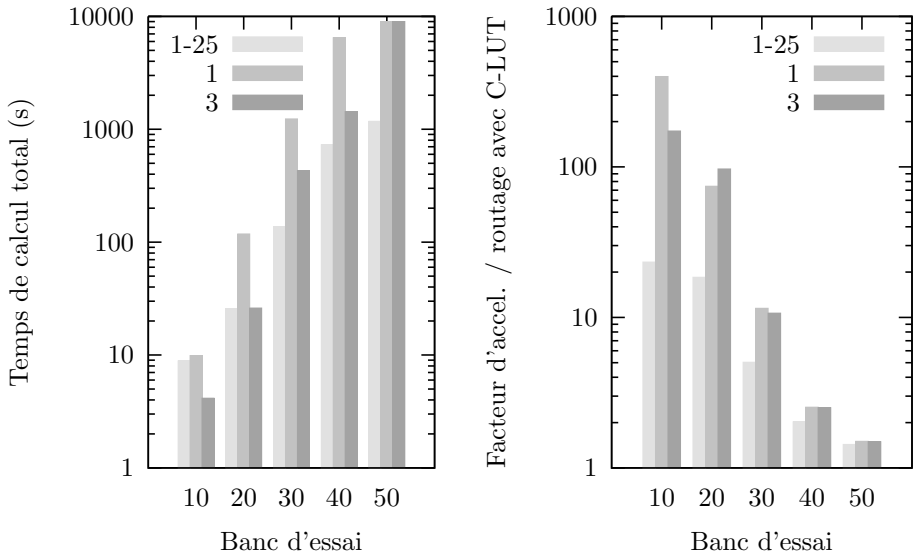


FIGURE 4.3.2: Temps de calcul et facteur d'accélération apporté par la nouvelle fonction de coût. L'accélération est normalisée par rapport à la meilleure solution, soit l'utilisation la table complète d'équivalences. Les temps de calcul pour les bancs d'essais attendus (entre 10 et 30 % de densité moyenne avec un équilibrage « normal ») sont routés en moins de 10 minutes, soit l'objectif pour le système WaferBoard.

4.4 Conclusion

Ce chapitre a présenté un nouvel algorithme de routage applicable aux réseaux d'interconnexions maillés. Cette contribution peut équilibrer les délais de plusieurs groupes de *net* de taille arbitraire. Les algorithmes proposés sont d'abord une extension du routeur récemment publié *RCV*, avec l'ajout d'une heuristique pour relaxer itérativement les bornes de délai sur chaque groupe de *net*, jusqu'à ce qu'une solution équilibrée soit trouvée. Cette extension non-triviale a montré qu'il était possible de réaliser un tel équilibrage. Une modification de la fonction de coût utilisée par *RCV* a été proposée, qui n'impacte pas la qualité du routage mais démontre une accélération très conséquente, jusqu'à un facteur 400 avec les bancs d'essai utilisés. Un autre algorithme destiné à accélérer les calculs, est basé sur une table de chemins équivalents. La technique de construction a été explicitée, et l'intérêt en terme de temps de calcul mesuré : la possibilité de coupler cette approche au routage par permutations la rend attrayante, avec un facteur d'accélération par rapport à un A^* qui atteint un maximum de 4.24.

Ces techniques mises ensembles, le routage d'un *netlist* attendu comme très dense pour l'application visée, soit un taux d'occupation de 30 % et équilibré de façon raisonnable, prend moins de 8 minutes de temps machine. Ce temps de calcul rentre dans les requis de l'applications, et le routage de *netlist* moins denses peut se faire en quelques secondes seulement. Ces contributions significatives rendent l'application du WaferBoard attractive pour le prototypage rapide.

Chapitre 5 - Article 2 - A PCB Netlist Model and Generator for a Routing Algorithm

Étienne Lepercq, Yves Blaquière, Yvon Savaria

IEEE Transaction CAD, soumis le 15 juillet 2012

5.1 Abstract

Benchmarks are widely used in the design automation community. Several benchmarks are available for comparing and validating VLSI and FPGA routing algorithms, but no netlist generators for PCB routing algorithms is publicly available. This paper proposes a netlist model based on a probabilistic method, a novel approach that can be used as a ground work to validate PCB routers. An analysis is presented for several industrial PCB netlists that we obtained as part of our work. It is shown that the Rent's plot and the derived Rent's exponent, used to model netlists for netlist generation in VLSI and FPGA, does not show a repeatable pattern from one PCB to another. Accordingly, a different model is proposed. This model is based on the known approach of net-degree and net-length probabilistic distributions, and refines the two distributions to better fit PCB profiles. It also uses a few user-specified parameters, such as netlist size and density. An algorithm that implements this model is proposed in this paper, which generates synthetic netlists to create PCB-like loads on a routing algorithm. The reported synthetic netlist generator is used to validate a routing algorithm for a recently proposed wafer-sized active integrated circuit capable of programmably interconnecting integrated circuits deposited on its surface. The resulting synthetic netlists as well as real netlists are routed, and results compared. Some 2 % difference on routing resources usage for real netlists and synthetic netlists are observed with less than 10 % difference in computation time for most netlists, which clearly confirms the validity of our model.

5.2 Introduction

In the electronic design automation (EDA) community, benchmarks provide a reference to compare, characterize, identify limitations and confirm capabilities of algorithms and CAD tools. In the VLSI design automation community, several public domain benchmarks are available and are widely used by academic researchers. For instance, IBM-PLACEv2, ISPD'07 and ISPD'08 [83, 13] have been used to validate VLSI placers and routers. As for PCB benchmarks for routers, there was PCB benchmark 2000. It has been used for the PCB 2000 conference West [109], but it is no longer publicly available.

PCB components have to be physically placed before routing, and no such PCB benchmarks with placed components are publicly available. The lack of public benchmarks for academic research has been reported in the past [85], and mentioned as the root of too few academic PCB routers being published, and of difficulties when comparing routing algorithms. Due to the very proprietary nature of netlists in the PCB industry, netlists (especially placed-netlists) are kept confidential. It is expected that this situation will not change in the foreseeable future, therefore a synthetic benchmark generator available for academic research is needed. Stroobandt *et al* [86] also confirmed that well-designed synthetic benchmarks provide control over important netlist characteristics (such as size, density, complexity, functionality). More importantly, each characteristic should be independent from each other, and reflect a realistic input for the tool studied. In this paper, we focus on generating realistic inputs for a family of routing algorithms that takes PCB netlists as input. This is necessary in a project that aims at implementing a complete rapid prototyping system platform [96]. This platform supports rapid prototyping of electronic systems and normally accepts user specified PCB netlists as input.

For netlist generation algorithms, the literature proposes two major metrics [110]: the Rent's exponent and the net-degree distribution. The Rent's rule [87] represents the relationship between the number of terminals T and the number of blocks B in a circuit (or partition), as a power law:

$$T = kB^p \tag{5.2.1}$$

where k is the average number of terminals per block and p is the Rent's exponent.

It was observed in the literature [111, 112] that this rule holds for any number of partitions in netlists that represent VLSI or FPGA circuits. The Rent's exponent provides a hierarchical measure of the interconnection complexity for a given netlist. It is generally admitted that the Rent's exponent is comprised between 0.5 and 0.75 [87] for the linear region of a Rent's plot. Three regions in Rent's plot have been proposed and analyzed thoroughly by Pistorius *et al* [9]. Regions II [87] and III [88] of Rent's plot are well-known deviations of the law, respectively for small number of blocks (region II) and large number of blocks (region III), where the actual number of terminals is generally smaller than what is predicted by equation 5.2.1.

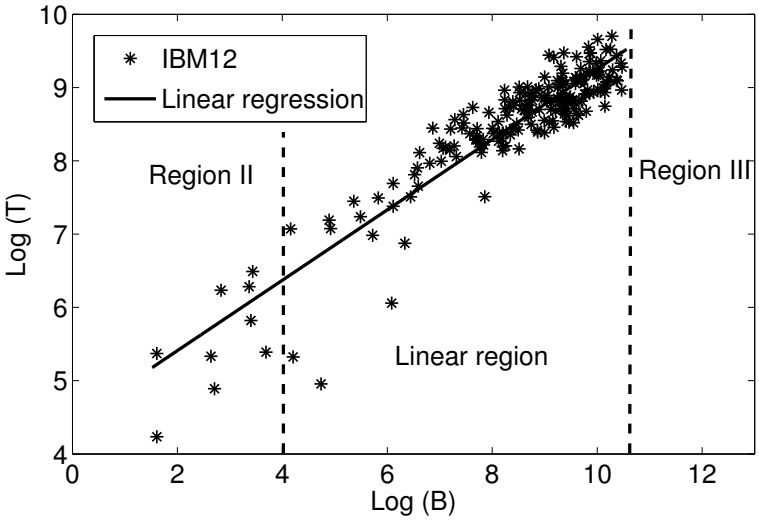


Figure 5.2.1: Rent characteristic of IBM12 benchmark (IBM-Place V2), a VLSI benchmark used for VLSI placer and global routers, created using the technique described in [9]. This netlist does not have data in Rent region III. The Rent exponent is evaluated to be 0.48 with $R^2 = 0.85$.

The Rent's plot can be created with the following simple algorithm : for each partition of the circuit, compute the number of blocks comprised in the partition, along with the number of terminals that are connected to terminals outside it. Historically, partitions were chosen by subdividing the circuit regularly in power-of-two rectangles. For each partition, a data point is generated for each pair (number of blocks, number of terminals) and plotted on a log-log graph. However, such method introduces a bias towards small partitions [9]. Indeed, many small partitions are generated, each having the same weight as larger - and rare - ones. The Rent's law supposes that partitions of any given size is related to another by the number of terminals and blocks in it; the method for extracting the Rent's exponent should use partitions equally distributed in terms of size and position in the design. Recently, a method for creating partitions less biased toward small ones was proposed by Pistorius *et al* [9]. It randomly generates partitions with various sizes and physical positions, and is used in this paper. A typical example of the Rent's plot that we computed from IBM place benchmark (VLSI circuit) using this method is given in figure 5.2.1. The power-law relationship is clear for the central linear region, which corresponds to region I. In figure 5.2.1, region II is also apparent below $\log(B) = 4$ whereas region III is less evident as the power law of region I remains an acceptable fit of the data up to $\log(B) = 11$, which includes all the data set.

The Rent's rule has been historically studied for PCB netlists many years ago [87], but recent research focused on FPGA and VLSI applications, as in [113, 114, 115]. Indeed, Rent's rule was used successfully for wirelength prediction [89, 90] mainly for guiding VLSI and FPGA placement algorithms. Rent's rule is reanalyzed in this paper in relation with various modern real world PCB netlists (digital, analog and mixed ones).

The net-degree (or fanout) distribution is usually formalized as an exponential-law [91, 92]. Some authors in

the literature have also analyzed several distribution models, such as a polynomial estimator in [93]. Unlike what was observed for VLSI and FPGA netlists, where about 75 % of nets are of degree 2 and 3 [116], we demonstrate in this paper that the net-degree distribution shows major differences for small net-degrees (typically less than 5) from one netlist to another. We propose a novel net-degree model to accommodate such disparity. This is significant as the net-degree distribution has an impact on local congestion, which tends to pose major challenges for routing algorithms [117]. Indeed, nets of degree 2, like busses, often create congestion due to a local peak density. The proposed netlist generator defines a set of parameters to help modeling busses in a more realistic way than other netlist models presented in the literature.

The net length distribution computed from rats nets (unrouted nets) is an important metric for the proposed netlist generator. The PCB EDA community studied the spatial distribution of rats nets to estimate the routability of a given placed netlist [118]. In general, rats nets length distributions are generated using a gaussian distribution [94] with a parameter that defines its standard deviation. However, this approach does not reflect reality for PCB routing algorithms, as shown in this paper.

PCB routing algorithms have to deal with a variety of component shapes, bounding-boxes that define specific routing constraints and allowances in sections of the PCB, as well as local routing rules that are derived from electrical properties of nets. In this paper, the targeted routing algorithm tackles a subset of the general PCB routing constraints: it supports virtually any PCB shape, is independent of the interconnection network used by the targeted routing algorithm, and accurately models wirelength and net degree distributions of PCB netlists. While it takes PCB netlists as inputs, it uses an interconnection network similar to those used in FPGAs rather than the typical routing model of a PCB. Therefore, this paper proposes models and algorithms that can be used as a good starting point for full-featured PCB netlist generator. Several possible extensions of the proposed algorithm for generating PCB netlists are discussed in section 5.6 to support other PCB routing constraints. Finally, the proposed model is directly suitable for the targeted application and represents a self-contained work on its own.

We claim in this paper the following contributions: (i) the first published algorithm for a PCB netlist generator, (ii) a novel model for net-degree distribution that closely matches the specific features of PCB netlists, (iii) a study of Rent's type modeling applied to modern PCB netlists showing that it does not fit well for real-world netlists. These observations lead to the conclusion that Rent's rule can hardly be applied to generate PCB netlists. We acknowledge the fact that the proposed PCB netlist generator is not complete for validating all features of state-of-the-art PCB routers, but claim that it is a good first step towards that goal.

This paper is structured as follows: in section 5.3 the analysis of Rent's rule, net-length and net-degree distributions for a subset of obtained real-world PCB netlists is presented. This section is by no means limited to the targeted application, and is based on a set of actual PCBs. The net-degree and net-length distributions extracted from real PCBs netlists are used to derive a first model for PCB netlist generation. The proposed netlist generation algorithm

is presented in section 5.4. This model is characterized in section 5.5, where routing results (indirect validation) for synthetic and real PCB netlists are reported and compared by using a specific routing algorithm, developed for the targeted application. The obtained results and algorithms are then discussed in section 5.6, to better understand the scope of this paper and how it would apply to PCB routing and PCB netlist generation. Finally, our main results are summarized in the conclusion of this paper.

5.3 PCB Netlist Model

The main purpose of our PCB netlist model is to serve at the core of a netlist generator that produces realistic loads to a routing tool under development. Routing algorithms typically use a graph to represent the interconnection network; the model is very generic, and composed of vertices that represent possible terminals positions (component pins or interconnect terminals) and each edge represents a wire that links two adjacent vertices. The model is independent of the grid density, which is defined with a parameter. The capacity of an edge is the maximum number of nets to which it can concurrently be assigned. In this paper, a route is defined by its source vertex and destination vertex (see figure 5.3.1). A path is a list of edges that connects two vertices (and thus two component pins) in the interconnection network. A path fully describes a route. There could be several possible paths for a route. These definitions are illustrated in figure 5.3.1, for a route from vertices v_s to v_d. Two different paths are shown, each with its own list of edges. Notice that the proposed model is agnostic regarding the topology of the interconnection network. The one used for routing generated netlists in this paper is described in section 5.5.

Therefore, a net in the netlist is a set of routes that belongs to the same electrical node. The interconnection network model supports any interconnection network topology.

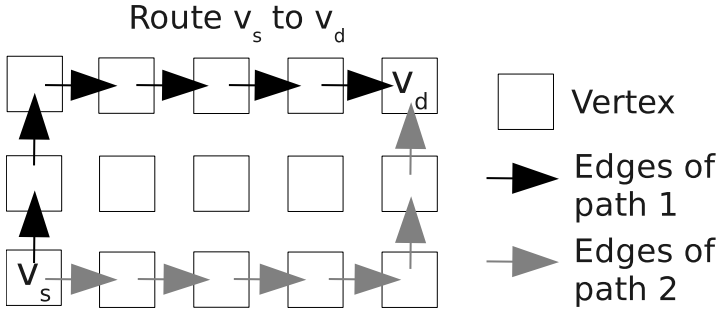


Figure 5.3.1: Two different paths (black and gray arrows) for one route, from vertices v_s to v_d. This example assumes that edges connect each vertex to its 4 nearest neighbours. Other graph topologies are possible, for example with non-Manhattan edges.

5.3.1 Model Parameters

The proposed netlist model is based on several parameters. A user parameter, the distance between two adjacent vertices in mm, allows to convert distances in the graph into physical distances. The physical length of a path in the graph can be retrieved by multiplying the distance between two adjacent vertices with the total number of vertices contained in the path. In other words, this parameter is the resolution of the grid when placed in the physical PCB. Consequently, the graph size defines the routing area and the total physical area of the netlist. The maximum number of pins per vertex in the graph supported by the technology is also defined as a parameter: in this paper, it is called an access point, and it defines a possible component pin position on the graph.

The pin density is a critical parameter and it depends on the proportion of available access points. The pin density is the probability for an access point to be used by a component pin, over the whole routing area. The pin spatial distribution associated to the pin density is the pin distribution over the routing area; the netlist model uses a 0-centred two-dimensional gaussian distribution truncated to the routing area.

The parameters related to the pin spatial distribution allow fine control over the net length distribution. Thus, netlists generated using the proposed model can represent realistic workloads for routing algorithms, as shown in section 5.5. The pin spatial distribution is described with a standard deviation. A larger deviation can reflect the possible difference between a larger density at the center of a PCB as compared to a smaller density at its periphery. The standard deviation controls the peak density. It provides a simple model that represents many typical PCB designs, where the center of the board is often used to place large ICs like FPGAs or microprocessors. The 0-centred gaussian distribution models *local* pin density at the center of the routing area that is larger than at the periphery.

Rent's rule is commonly used as a reference for generating synthetic VLSI or FPGA netlists because it captures the hierarchical interconnection complexity in a netlist [86]. As shown in section 5.3.2, the Rent's rule does not accurately reflect the wiring density found in PCB netlists. The proposed netlist model is rather based on net-degree and net-length distributions. These two distributions of commonly used parameters are modeled according the observed PCB distributions. In addition, we propose two new parameters for allowing user to generate various profiles: for each desired net-degree, one can define the minimum and maximum occurrence probability. These parameters offer more flexibility and are proposed to fit the variety of observed profiles of actual PCB netlists (see section 5.3.2). An algorithm that can generate netlists with profiles more similar to real PCB netlists is also proposed based on these distributions. Moreover, our netlist model takes into account busses that include parallel nets due to smart pin placement and delay constraints. Parameters introduced to allow modeling busses are the proportion of nets that belong to a bus, and the distribution of bus width. The bus width relates to the number of electrical nodes that belong to the same bus, and that should preferably be close together. The bus width distribution in the model is a gaussian centred around the average bus width, and it has an associated standard deviation.

5.3.2 Rent's Rule Analysis for PCB Netlist Generator

The Rent's rule is a commonly used model in netlist generators for VLSI and FPGA. This section shows that this model cannot be used in a realistic PCB netlist generator. The proposed analysis and conclusions are entirely independent of the specific application for which we created this model, and it can be also used for research related to true PCB routers. Our work is based on statistics extracted from twenty real PCB netlists, from which sixteen are placed. The produced Rent's graphs depend on the partition technique used. The graphs were drawn for each netlist with three different partitioning techniques. The first one is a technique for computing the Rent's exponent based on [9], which is more recent, and which has the advantage of not being biased towards small partitions, in contrast to the historic method that generates power-of-two rectangular partitions. In first technique proposed in [9], partitions are square with randomly generated center position and side length. The square can be made rectangular if a generated partition overlaps the boundaries, and rectangles with aspect ratios larger than 4 are discarded. We also used the described filter for Region III partitions [9]. This filter rejects all partitions found to contain more than 1/3 of the total number of chips (or blocks), enforcing their re-generation. As a first step, the analysis is conducted assuming each chip is one component that counts for one block (B=1) in a partition irrespective of its complexity. Two other methods that accounts for the different complexity of various components are considered later.

For example, figure 5.3.2 shows Rent's plots of two representative real PCB netlists, with a linear fitting for PCB1. The partition approximately located at $\log(B) = 1.75, \log(T) = 1.5$ on PCB1 plot was excluded from the fit, as it is clearly outside the main cluster. Partitions located above $\log B = 5$ present a pattern similar to that found in Region III with VLSI or FPGA netlists, even if netlist partitions were created according to Pistorius *et al* [9]. Such patterns should be excluded from the fit. Region III tend to appear for relatively large partitions in PCB netlists when compared to VLSI or FPGA ones (see [9] for example). Accordingly, the fitted line on PCB1 in figure 5.3.2 excludes those data points. That fit corresponds to a Rent's exponent of 0.18 with $R^2 = 0.34$, which is a low correlation. These results are very different from those obtained with VLSI and FPGA netlists, where the Rent's exponent is typically comprised between 0.5 and 0.75, and the R^2 is usually close to 0.85 [86], as in the example in figure 5.2.1). A Rent's exponent close to 0 corresponds to a very simple netlist, such as a shift register. The small Rent's exponent extracted from figure 5.3.2 seems to imply that components on the analyzed PCB are relatively less interconnected, as compared to VLSI and FPGA netlists. This may be a consequence of the high integration level of components found on today's PCB (especially digital PCB), with few components often interconnected with wide busses. Thus, the observed netlist interconnection complexity, as measured by the Rent's exponent, is much lower than with VLSI and FPGA circuits. On the twenty real PCBs analyzed, about 60 % show such profile, with a very flat Rent's plot on region II and a low R^2 when a linear fitting is applied.

About a third of the analyzed PCBs present a different profile. An example of a Rent's plots for a PCB provided by the same designer as the one above is shown in figure 5.3.2 (PCB2). It would be hazardous to try any fitting on such

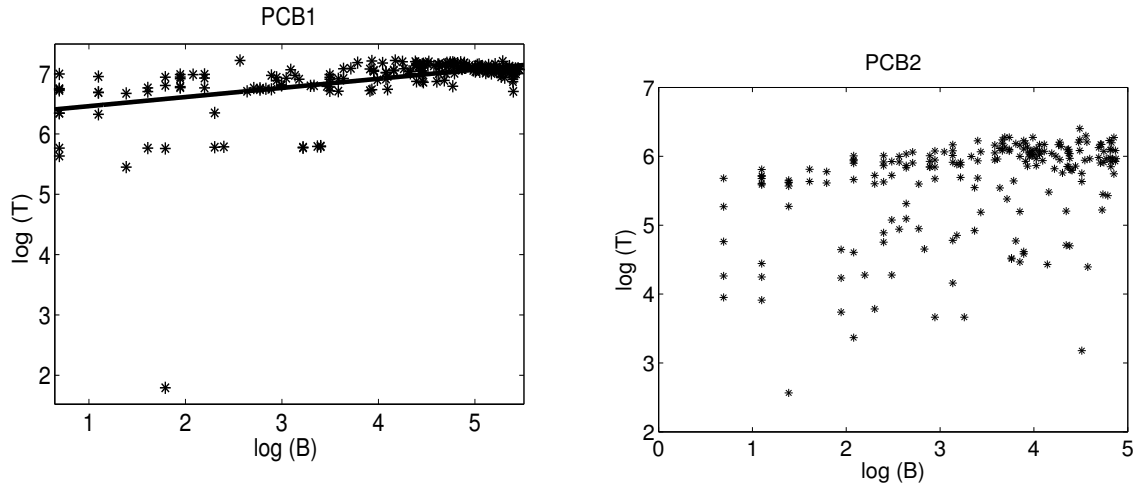


Figure 5.3.2: Rent's plot for two representative PCB netlists, PCB1 and PCB2. PCB1 above presents a better fit (still not good) when compared to others, while PCB2 below is one that seems incompatible with Rent's power-law (as is 30 % of our data set) .

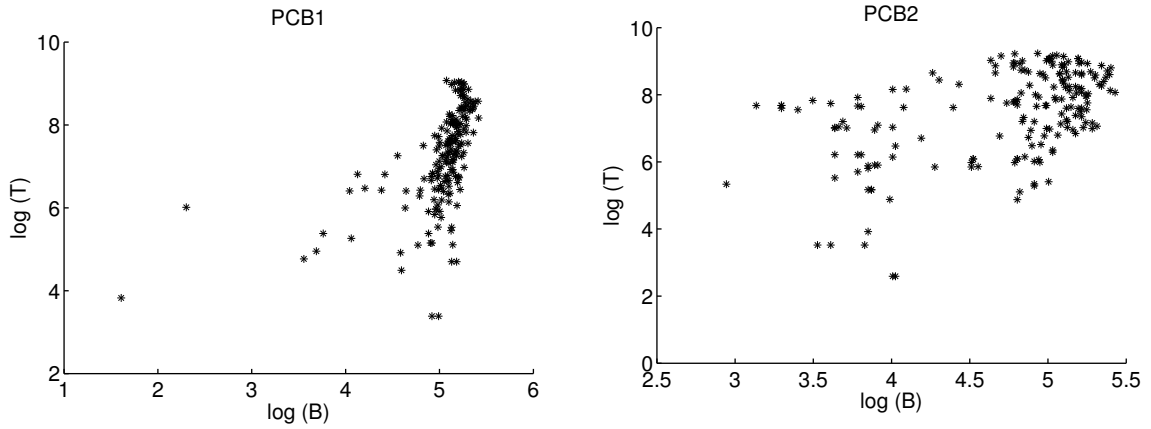


Figure 5.3.3: Rent's plot of PCB1 and PCB2 using random square as partitions, and plotting the total area of chips inside the partitions. These plots are representative of the available netlist set, and they cannot be modeled using the Rent's power-law.

data. Such difference cannot be attributed to a badly designed board, as it has been made by the same designer.

As a second experiment, the same partitioning technique was used, but the area of chips inside the partition is plotted, rather than the number of blocks. The same two PCBs are plotted using this technique on figure 5.3.3. The respective profiles are similar to those obtained with the previous technique. As expected, there is in general a proportionality relationship between the number of pins and the occupied area. The conclusion one can draw from these results is identical to that obtained with the previous partitioning technique, i.e. the Rent's law could not fit the considered modern PCB data sets.

To validate these observations, a third partitioning technique was used. A partition is assumed for each PCB component. The area of a PCB component is used as the number of blocks B it represents, in square millimeters, and

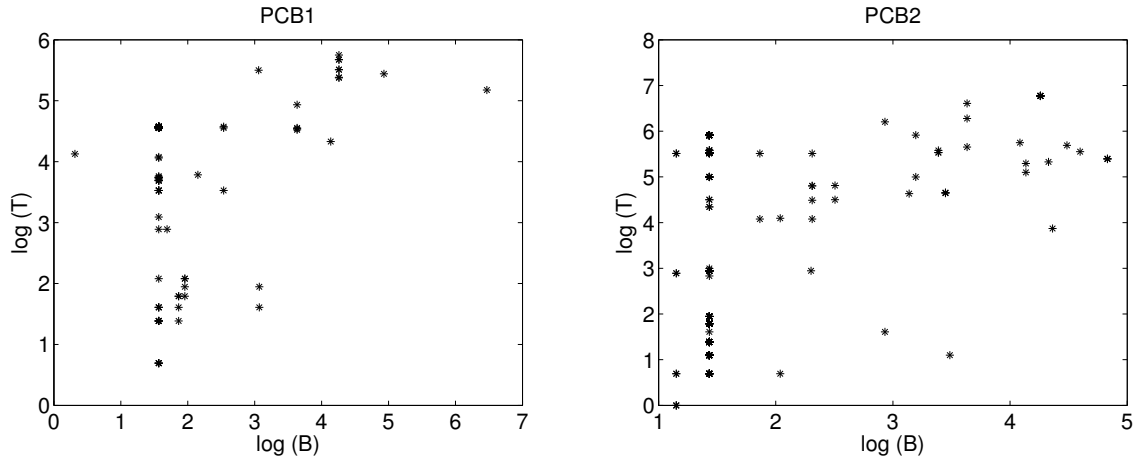


Figure 5.3.4: Rent's plot of PCB1 and PCB2 using each component area on the board as a partition of the netlist. These plots are representative of the available netlist set, and they cannot be modeled using the Rent's power-law.

the number of terminals connected to external nets is used for T. Rent's plot generated from these assumptions were also used to determine if a general model of PCB components can be extracted from PCB netlists, and used to generate synthetic netlists. figure 5.3.4 shows two representative Rent's plot. Notice that these plots take into account all PCB components, such as resistors, capacitors, connectors, IC, etc. The data points shown on the plot represents the number of different components and not the number of components because several components could have the same B,T pair.

While the largest components have the largest number of pins connected to others (top-right corner), the Rent's law can not fit these data sets with any accuracy. Linear regressions on these data presents low $R^2 < 0.3$.

We concluded from experiments conducted on the 16 placed netlists that an accurate model cannot be extracted using the Rent's rule on most of them, and the Rent's exponent is not used in our proposed netlist generator. Our novel model described in the following section better fits the specific needs of PCB netlist generation. Our model is based on net-degree and net-length distributions (rats nets).

5.3.3 Analysis and Generation of Netlist Degree and Net Length Distributions

The net degree (or fanout) and its distribution over all nets were extracted from analysis of twenty real world PCB netlists ; the OpenMoko's mobile phone design [119]; an high-speed industrial design named ID and twelve other PCBs from the same company, plus a few simple PCBs. All PCBs have between 2 and 14 metal layers (most have 4-6). They comprise from 76 to more than 11,000 nets and from 470 pins to nearly 40,000 component pins.

The netlist degree distribution of a representative subset of these designs is shown figure 5.3.5. The netlists show similar net degree distribution for net degree larger than 4. However, around 75 % of the nets in the ID design are of degree two, while PCB7 has less than 10 %, and some are closer to 20 %. The literature in general uses an exponential model for the net degree distribution. However, the PCB netlist model has to take into account the large disparities of net degree in PCB netlists, especially for small net degree. The net degree distribution is built from a user defined

profile. The profile describes the minimum and maximum occurrence probability for each user defined net degree (typically for net degree 2 to some user defined maximum). For example, Table 5.4 shows the parameters that can be used to generate a profile similar to the ID design. It is possible to build an exact profile when using equal minimum and maximum values for a same net degree, but the sum of all minimum probabilities must be equal or less than 1. Indeed, the random generator expects the sum of probabilities to be 1: the generated profile will be distorted otherwise. In our netlist generator, if this sum is less than 1, the net degree distribution builder will randomly choose a net degree and increase its probability. This increase is randomized between the actual value of the net degree and the maximum assigned to the profile. Such process is repeated until the sum of the probabilities for the generated profile is exactly equal to 1. One may note that if the user defined profile (minimum and maximum for each net degree) is wrongly defined (e.g. minimum and maximum are all equals, and the sum is less than 1 for all net degrees), the profile builder may generate a distorted profile.

With this technique based on netlist degree distribution, it is also possible to define one single profile that could be used to randomly generate realistic net degree distributions. Indeed, one can provide a user profile that has minimum and maximum probabilities for each net degree. Tight control on the generated profile can be achieved by providing more constrained minimum and maximum probabilities.

The distribution of the net length, measured as the physical distance between two placed component pins, is another critical parameters of our PCB netlist generator. The two-pin length distribution of the same PCB netlists are shown in figure 5.3.6. Contrary to net degree, the net length distribution of all netlists are much more similar as shown in that figure. The netlist generator uses the same distribution function for all netlists, namely, the sum of two exponentials. The extracted best fit parameters are $a = 0.81$; $b = -16.61$; $c = -0.005$; $d = -8.47$. A Chi-2 test with 95% confidence accepts the model across all analyzed PCB designs. It is observed that when the method used to place components reduces the global distance between pins, the net length distribution is usually steeper, i.e., the distribution contains a large number of short nets. Typically, netlists with a large proportion of short nets are easier to route, and can be considered as well placed netlists. Accordingly, figure 5.3.6 depicts two examples of different profiles generated for an hypothetical placed netlist that can be generated with our netlist generator. The “well placed netlist” has the steepest

Table 5.4: An example of net degree distribution profile that could match the ID design. Similar distributions can match any profile.

Net degree	2	3	4	5	6	7	...
Min. occurrence probability	0.70	0.23	0.03	0.01	0.01	0.01	...
Max. occurrence probability	0.80	0.25	0.1	0.05	0.02	0.01	...

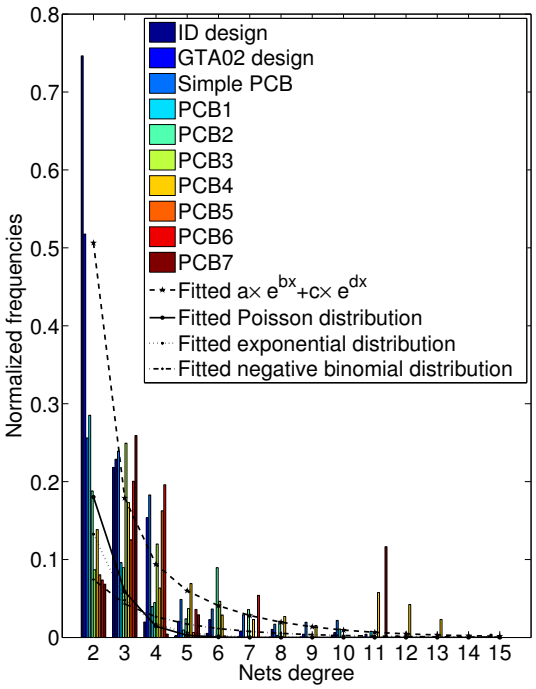


Figure 5.3.5: Histogram of the net degrees for several real netlists, and fitting with some distribution models using nonlinear least squares method and a 95% confidence rate. The exponential relation provides the best fit on all PCB profile (with various values), confirmed by a Chi-2 test with 95% confidence.

Table 5.5: Parameters used for netlist generation in this paper.

Parameter	Name	Value (min, max)	Standard deviation
X/Y axis Graph size (vertices)	Gx/Gy	(200, 400)	-
Pin average density (%)	Pd	(10, 80)	1
Busses proportion (%)	Bp	20	-
Busses average width	Bw	8	2
Average net length (mm)	W	(1, 200)	-
Average net degree	Nd	2.5	-

slope and it contains a large proportion of short nets (good locality), while the “badly placed netlist” represents a netlist with a larger number of long nets. These curves demonstrate that the proposed model allows fine control over the net length distribution to produce a wide range of benchmarks, which is highly desirable feature for characterizing routing algorithms. The parameters used to generate the netlists in this paper are summarized in table 5.5, along with their chosen value. However, all of these can be modified to build other profiles, as needed.

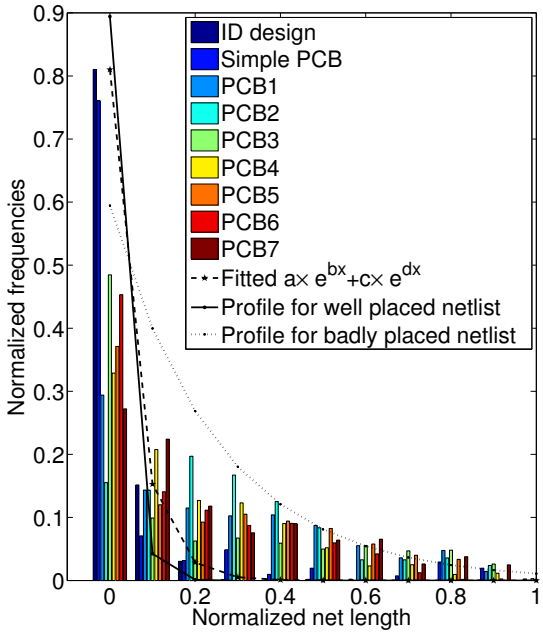


Figure 5.3.6: Histogram of net length distribution for PCB and VLSI netlists, fitted with an exponential distribution model. The fit is accepted by a Chi-2 test with 95% confidence. The net length distribution is normalized over the size of each netlist routing area.

5.4 Netlist Generation

The proposed netlist generation algorithm is suitable for the targeted application, a rapid prototyping platform for electronic systems recently proposed by Norman *et al* [96]. This prototyping platform is based on a wafer-sized CMOS circuit that includes a configurable interconnection network (see figure 5.4.1). This network links a large number of contact points (called NanoPads), each one being configurable as an I/O signal or as a configurable power supply. The 4×4 NanoPads are grouped in a simple building block called a cell, where each one supports up to two IC pins, named "access points". Cells are tiled within a reticle-image and the wafer surface is built from photo-repetition of such sea-of-cells reticle, with inter-reticle stitching for connections between reticles. A user simply places integrated circuits (ICs) on the active surface and an array of NanoPads detect the IC pins. ICs are then programmably interconnected through the defect-tolerant network (described in section 5.4.1) and the system is ready to run. Signal integrity is maintained by internal repeaters, mitigating crosstalk and attenuation, and ensuring a propagation delay mostly proportional to the wire length. Therefore, with this target rapid prototyping platform, several important constraints in PCB routing no longer exists, such as the variety of routing rules and the blocking areas (the routing area is homogeneous), while propagation delays present a major challenge for the routing algorithm of our target platform.

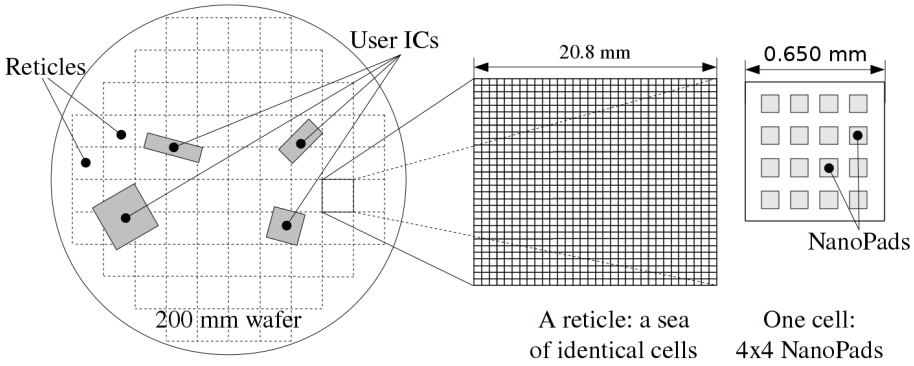


Figure 5.4.1: The prototyping system schematic: user's IC are manually deposited on the wafer's surface, external connections are possible as well as probing digital on-wafer interconnection signals – in real time.

5.4.1 Interconnection network description

The interconnection network embedded in the targeted application is a multi-dimensional mesh. This contrasts with the PCB or VLSI routing problems that can be modeled with a simple mesh, where each vertex is fully connected to its direct neighbours. The interconnection network is defined as a regular grid of crossbars, where each crossbar is modeled as a vertex in the interconnection network. For each vertex v, let us define a set S_v of distant neighbours in the four directions: the distance between v and its neighbours is expressed by 2^d, $d \in [0;D]$. D is the dimension of the network and is a positive integer. These distances are power-of-two, and an edge is a wire that links v and a vertex in the set S_v. Such construction is depicted in figure 5.4.2, where direct connections (i.e. wires) from the vertex at coordinates (0;0) to its neighbours at distance $2^0, 2^1, 2^2$ are shown. These wires exist for each vertex in the interconnection network. D, the dimension of the network, is sufficient to define the length of the longest wire. For example in the network of figure 5.4.2, $D = 2$ with the longest wire being $\lambda_{max} = 2^D = 4$. Repeaters are placed on long wires to ensure a wire delay being proportional to its length, to compensate for the attenuation in a standard CMOS process. The hardware being implemented, and for which the targeted routing algorithm is developed, has currently 6 different lengths of links that correspond to $L = 1$, 2, 4, 8, 16, and 32. Each vertex is physically called a cell, which size is 650 μm. However the algorithm is using a user parameter for describing the grid, and can be modified as desired.

5.4.2 Netlist generation algorithm

The netlist generator is shown in Algorithm 5.5. The netlist tool adds pins and nets until the total pins density *curPd* reaches the user specified parameter *Pd*. For each net, the net degree *Nd* is randomly generated using the distribution model proposed in the previous section. For instance, suppose the net degree generator is given distribution parameters of table 5.4. At initialization, the net degree distribution is generated according to the maximum and minimum occurrence probabilities for each given net degree. From table 5.4, one possible net degree distribution could be the one described in table 5.6. The detailed mechanism is described in section 5.3.3.

Table 5.6: An example of a generated net-degree distribution, that complies with profile of table 5.4.

Net degree	2	3	4	5	6	7
occurrence probability	0.73	0.23	0.08	0.02	0.01	0.01

Following initialization, each pin of a net is randomly placed using the following algorithm. A pin pair is placed along a virtual line (a fly route). First, the algorithm generates the line center position (*CX* and *CY*) according to the spatial gaussian distribution. Then, the wire length *wl* is generated according to a normalized wire length distribution (as shown in figure 5.3.6). The distribution uses normalized length, therefore the current wire length is scaled according to the user parameter of average net length. This user parameter has to be set in relationship with the routing area (see table 5.5). From these generated parameters and the orientation of the virtual line, two pins can be generated if they fit inside the routing area. The function verifies such property, and if false, it regenerates a set of pins. Therefore, the routing area may not be rectangular, as shown in figure 5.6.1 page 113. Indeed, a user passes as a parameter the graph to be used, which defines the shape of the routing area. For a net with an odd degree (for example, 3), the last pin recorded is the first of the generated pair, the other being discarded. As many pins as required by the current degree of the net are positioned and recorded using this pin placement algorithm.

Each generated net has a user specified probability of being a bus. In this case, the first pin pair is placed using the algorithm 5.5. Then, the getClosePin() function returns a pin whose distance to the previous pin of the bus is computed using twice the average density parameter, bounded to the maximum density permitted by the technology. If such position is already occupied, the algorithm simply chooses the closest position still available (i.e. where no pin is already recorded). Recall that when routing, priority of busses over other nets should be decided by the router, and are independent from the netlist itself. There is no need for a priority parameter in the netlist generator.

The position of the pin in a bus is determined so that all nets of the bus form a virtual line, so that it matches locality

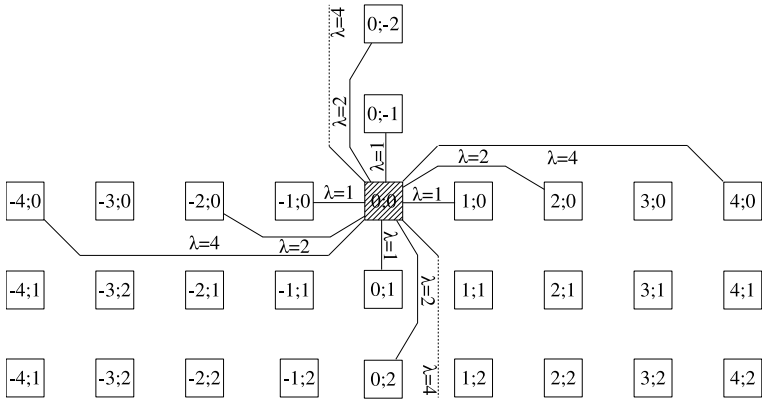


Figure 5.4.2: Multi-dimensional mesh network: vertices are represented by squares (physically, crossbars) with their relative coordinates, and edges as wires. Edges for the vertex at coordinates (0;0) are shown for a mesh of dimension 2 ($2^2 = 4$).

of most busses in PCB. The function getParallelWire() returns such result, by using the previous pin position and the orientation of the bus previously generated. Therefore, busses are placed so that unrouted nets create parallel lines (fly routes), close to each other, until all nets of the bus are placed. These two additional spatial constraints create local zones with high densities, and matches our observations on real PCBs where busses are often spatially grouped. An example of such generated bus of width 4 is presented in figure 5.6.1 page 113 on the zoom section. Notice on this figure that X and Y axis are not using the same scale: pins along the X axis are at maximum density (one at each access point) but for readability pins are represented with some distance.

When placing pins, the algorithm checks that the occupancy of the chosen vertex is not larger than its capacity (allowed by technology), and it regenerates a position otherwise. When the required pin density gets close to 100 % of the total capacity offered by the technology, the probability of generating a pin position that is not yet occupied dramatically decreases. Therefore, the total run time increases consequently. While netlists with average pin density close to the maximum offered by technology are unlikely, we have implemented another generating flow to handle such cases in a computationally efficient way. Indeed, when the total pin density gets higher than a threshold set to 90 %, pins position are randomly generated from the list of available vertices. This secondary flow was effective at keeping run times low, even in the case of very high densities. It was observed that it introduces a small acceptable bias towards long net length for a small proportion of nets.

An example of pin positions generated for the targeted application (a non-rectangular wafer-scale form-factor) is shown in figure 5.6.1 page 113. While it represents a low-density netlist (5% on average), there is still high local density that can be seen, as is often the case in a real PCB. If it is true that the stochastic approach adopted by the netlist generator clearly places pins in a non-meaningful way: results in section 5.5 will demonstrate that it still represents a very accurate load for the targeted routing algorithm.

5.5 Results

This section presents a validation of the proposed netlist model. This is done by comparing routing results of real netlists to the routing results for netlist generated using the proposed model. From the 20 available real netlists, only 14 were complete designs that contain placement and routing information : thus, only those 14 real netlists are compared to synthetic ones in this section. Each synthetic netlist is generated for the same set of input parameters defined above according to the physical size of the PCB and its average pin density. We used an in-house routing algorithm developed for the interconnection network described in section 5.4.1.

First, for each real PCB netlist, a synthetic netlist that matches its size and its average pin density is generated. Metrics were set to values reported in Table 5.5 for all netlists. The net-degree distribution profile that was used is reported in Table 5.7. This profile is able to generate any PCB netlist similar to those reported in figure 5.3.5. Secondly, each real PCB netlist and its associated generated netlist were routed using our routing algorithm. The percentage of

Algorithm 5.5 The netlist generation algorithm.

```
1:  procedure NETLISTGEN
2:      gaussianGen ← init(routingArea)          ▷ Init. gaussian random generator for pin position generation
3:      wirelengthGen ← init(avgNetLength)       ▷ Init. random generator for net length distribution
4:      netdegreeGen ← init(distribution)        ▷ Init. random generator for net degree distribution
5:      while pin density curPd is less than Pd do
6:          Nd ← getNetDegree()                  ▷ With appropriate distribution
7:          isBus ← getIsBus()                   ▷ Probability user specified
8:          orient ← random(0, π)                ▷ Orientation of route
9:          busWidth ← getBusWidth()
10:         curNet ← getNetName()
11:         while curNd < Nd do
12:             CX ← gaussianGen()               ▷ X pos. of center of net
13:             CY ← gaussianGen()               ▷ Y pos. of center of net
14:             if isBus == false then
15:                 orient ← random(0, π)        ▷ All nets of a bus have same orientation
16:             end if
17:             posPin ← getPosPin(CX, CY)       ▷ With appropriate wire length distribution
18:             pin ← recordPin(posPin, curNet)
19:             curNd ← curNd + 1
20:             if isBus == true && isEven(curNd) &&
            curNd < busWidth then
21:                 curNet ← getNetName()
22:                 posPin1 ← getClosePos(posPin, orient)   ▷ Bus wires are generated close from each other
23:                 posPin2 ← getParallelWire(pin)          ▷ Bus wires are parallel before routing
24:                 pin ← recordPin(posPin1, posPin2,
        curNet)                                  ▷ Places two new pins in bus
25:                 curNd ← curNd + 2
26:             end if
27:         end while
28:     end while
29: end procedure
```

Algorithm 5.6 The algorithm for generating pin positions according the wire length and spatial distribution

```
1: procedure GETPOSPIN(CX, CY, orient)
2:     wl ← wirelengthGen(CX, CY)                 ▷ Never generates outside routing area
3:     {X1, Y1, X2, Y2} ← getPos(CX, CY, wl, orient)   ▷ Retrieves start and ending points from line parameters
4: end procedure
```

wires used for each wire length after routing is stored for each netlist. The difference of wire usage between synthetic and real netlists are reported in Table 5.8. It shows that the routing resources consumption of synthetic netlists on the multi-dimentional mesh network is very similar to that of real PCB netlists and the observed differences are always lower than 2 %.

Figure 5.5.1 presents the relative difference between synthetic and real netlists for each wire length of the target technology. For each wire length, the plotted value is the average of the 14 analyzed benchmarks. The relative difference is small for length 1, 16 and 32, and they are always below 12 % for all tested netlists (figure 5.5.1). On average, the difference is about 6 % for interconnects of length 2, 4 and 8, and below 2 % for other resources.

The computation times to route real and synthetic netlists with the same routing tool were also extracted (table 5.9).

Table 5.7: Net-degree distribution profile used for generating netlists. This profile can match the various profiles that have been observed on real-world netlists.

Net degree	2	3	4	5	6	7	8	9...23
Max. occurrence probability	0.75	0.25	0.19	0.1	0.05	0.03	0.02	0.02
Min. occurrence probability	0.25	0.1	0.06	0.05	0.04	0.03	0.02	0.01

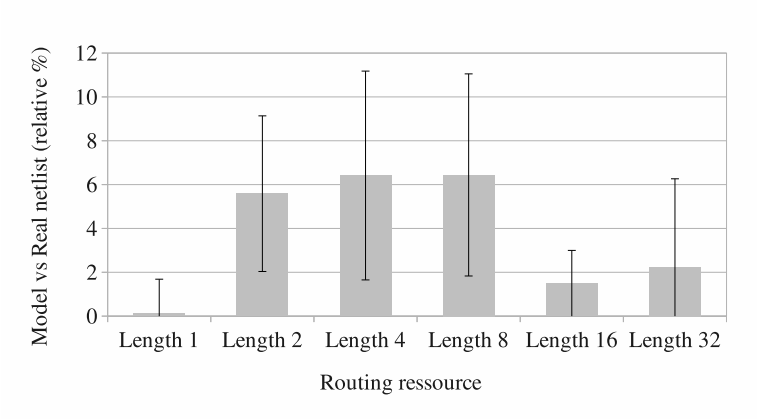


Figure 5.5.1: Relative differences between synthethic and real netlists (%) for each category of routing resources, averaged over the fourteen netlists. The differences are larger for interconnect lengths 2, 4, 8 but are always below 12 %.

It provides a good measure of how hard it is to route a netlist, for a specific routing algorithm. Results are the average of three routing runs for each netlist, and they were found to be similar for most netlists. Most relative differences are within 10 %. Only PCB1 and PCB7 netlists show larger differences: 22 % and 58 % respectively. Analysis of these cases showed that the net density of the these real benchmarks is much lower than the others, and they are not typical of industrial netlists. From these experiments, we conclude that the difficulty of routing the generated netlists is close to that of routing real designs. Notice that using a different router will show different routing times and/or routing resources usage than that we obtained. The important result is that real versus synthetic netlists show comparable profiles once routed, and this expected to be an invariant among routing algorithms.

5.6 Discussion

The proposed algorithms have been applied to a specific application, a novel prototyping platform for electronic systems. In essence, it can be considered as a programmable PCB that can support any components and take as input standard PCB netlists. Therefore, our work is tightly connected to the PCB world. However, several constraints of PCB routing have not been considered in this paper and are not required by the targeted application. Among them, the

Table 5.8: Percentage of routing resources used for real netlist versus generated one (for several wire lengths of the interconnection network). Occupancy percentage are very similar for real and synthetic generated netlists.

Netlist	Length 1	Length 2	Length 4	Length 8	Length 16	Length 32
ID	-0.01	0.26	0.2	0.26	0.11	1.62
Simple PCB	0.00	0.00	0.00	0.00	0.00	0.00
PCB1	0	0.32	0,.23	0.3	0.08	0.26
PCB2	-0.05	0.01	0.05	0.05	0.00	-0.15
PCB3	-0.04	0.05	0.06	0.06	0.01	-0.07
PCB4	-0.06	0.05	0.04	0.05	-0.01	0.11
PCB5	-0.01	0.02	0.01	0.02	0.00	0.01
PCB6	-0.01	0.03	0.02	0.03	0.00	0.59
PCB7	-0.05	0.06	0.05	0.07	-0.01	0.03
PCB8	-0.05	0.06	0.07	0.08	0.03	0.03
PCB9	0.00	0.06	0.06	0.07	0.04	0.00
PCB10	-0.01	0.06	0.06	0.09	0.03	0.06
PCB11	-0.03	0.08	0.06	0.1	0.00	0.02
PCB12	-0.04	0.09	0.07	0.12	0.02	0.05

most important ones are floorplan modelling and electrical characteristics of nets from which local routing rules are derived. This section discuss how the proposed work can be extended to model these aspects of PCB routing.

In most PCBs, nets will not necessarily share same electrical properties: some are power nets that requires large traces, or have constraints regarding electromagnetic interferences (EMI), or signal integrity constraints (IR-drop). From these typical PCB netlist parameters, specific routing rules are derived for the routing algorithm. One solution to extend the proposed algorithm is to associate to each net a set of realistic electrical parameters chosen from a user-defined list. Such list describes the various electrical properties that exists for the generated netlist. A model representing the distribution of these properties should be build, and a generator that can apply these properties to each generated net should be developed. This can be added at line 18 of the algorithm 5.5, in the recordPin function. In the targeted application, these constraints are not handled by the routing algorithm, as signal integrity is maintained internally by repeaters embedded in the wafer.

From the routing algorithm perspective, the floorplan of a PCB consists primarily of component placement, and specifies bounding boxes that restrict routing area for a set of nets: these rules are derived by the user from the PCB shape, as well as mechanical and thermal constraints. The proposed model takes an interconnection network as input, and places pins inside the corresponding area (function getPosPin line 17 of algorithm 5.5). Therefore the algorithm handles virtually any PCB shape specified by the user through the interconnection network description. Handling

Table 5.9: Average computation times to route real and synthetic generated netlists for three routing passes. Most synthetic netlists generated with our tool are slightly harder to route than real ones, but the differences are in general within 10 %.

Netlist	Computation time (s)		Relative difference (%)
	Real netlist	Gen. netlist	
ID	38.03 ± 1.07	39.21 ± 1.51	3.10
Simple PCB	1.01 ± 0.11	1.00 ± 0.13	1.00
PCB1	15.24 ± 0.27	18.53 ± 0.15	22.48
PCB2	1.47 ± 0.06	1.45 ± 0.05	1.38
PCB3	2.79 ± 0.07	2.90 ± 0.10	3.82
PCB4	6.59 ± 0.02	6.57 ± 0.30	0.29
PCB5	25.38 ± 0.70	25.45 ± 0.87	0.28
PCB6	26.33 ± 0.17	28.73 ± 1.22	9.10
PCB7	6.86 ± 0.15	2.86 ± 0.13	58.25
PCB8	3.12 ± 0.09	3.09 ± 0.19	1.13
PCB9	4.68 ± 0.29	4.86 ± 0.20	3.95
PCB10	6.29 ± 0.11	6.61 ± 0.16	5.05
PCB11	6.92 ± 0.17	7.32 ± 0.21	5.79
PCB12	25.32 ± 0.37	25.12 ± 1.20	0.77

component floorplan requires a component model, while the proposed algorithm places each pin pair independently from others using a stochastic approach. A possible solution is to build a component generator using a model: PCB component are mostly regular, and component families can be extracted from PCB netlists. A typical PCB uses a small proportion of large, central components like BGAs, surrounded by passive components (resistors and capacitors). Connectors and medium-sized mixed analog and digital signals usually constitutes the remaining of such typical PCB netlist. A model of the components and their frequency of occurrence for typical PCB would need to be built from analysis of netlists, and a stochastic generator should be built from this model. To generate netlists using a component generator, components should be generated before any net generation, as a PCB designer works (for example, between line 4 and 5 of algorithm 5.5). The number of components should be bounded according the user-defined total number of pins (derived from average pin density), and placed according to the shape of routing area. A placement model can also be built from analysis of real PCB netlists. Once components are placed, net generation described in algorithm 5.5 can be used with following modification. The orientation of nets as well as calls to gaussianGen() (lines 12-15) should be discarded. The getPosPin function should choose a pin pair best suited to fit the generated wirelength. Finally, getClosePos and getParallelWire should also choose from previously generated component's pins instead of placing them. The rest of the algorithm remains unchanged. Such component model was not designed and integrated because the proposed approach generates a realistic load for our routing algorithm, and is simpler than creating a component

model.

One important aspect of this work is that it models typical PCB: it is generally admitted that PCB netlists present a very large spectrum of shape, embedded technologies, and applications. A PCB netlist model would produce accurate results for the family of studied netlists, and non-typical examples will always exist: the purpose of a netlist generator is to generate a realistic load for routing algorithms, and typical PCB netlists provides a desirable input for such task.

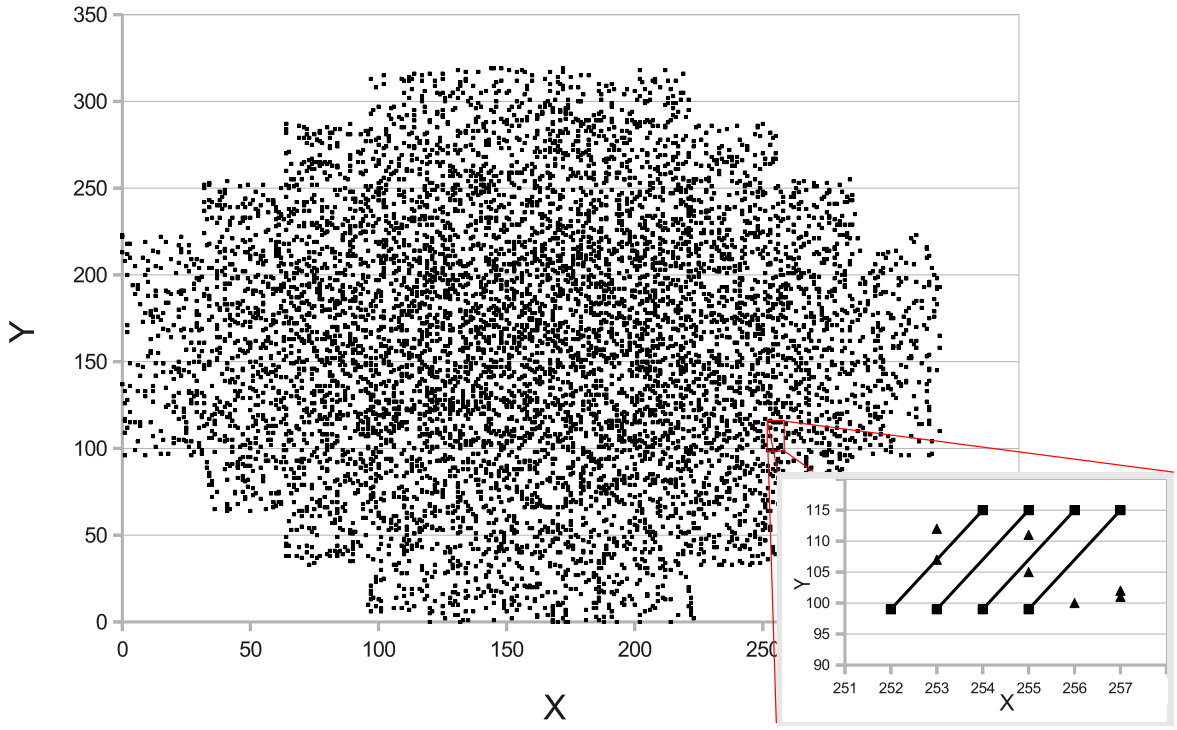


Figure 5.6.1: An example of pin positions as generated by the proposed algorithm (dots), along with a zoom on a generated bus positioned in the rectangle (252;115) (257;115). The average density is only 5 %, and the form factor is the one of the targeted application. It is not rectangular, as can be infered from the dots, but a rectangular shape benchmark can also be generated. Local density in some places is high, as would be expected in a real PCB. The fly lines on the zoom section represents connectivity between pins. Other pins represented as triangles have been placed by the algorithm in the area.

5.7 Conclusion

In this paper, a netlist generator suitable for building a realistic load for a routing algorithm that takes a PCB netlist as input has been proposed. The generated netlists were compared to different real PCB netlists of various sizes and purpose. The generator is parametrized with several statistics, such as net degree distribution, pin density, net length distribution. We also showed that the Rent's rule does not fit modern PCB netlists, a very different conclusion from the one that was reached for VLSI and FPGA circuits. A net degree distribution model and associated generation

algorithm were proposed to accommodate a wide range of profiles that can match observed profiles. This model and the associated generation algorithm provides a generic solution for generating representative net degree distribution. This model can match the studied characteristics of real industrial PCB netlists. The algorithm provides the ground work and infrastructure for a first set of important PCB netlist parameters, the net-degree distribution, wirelength distribution, average density and various shapes of routing area. The algorithm and model can be extended to take into account mixed routing rules, electrical characteristics. However, the proposed model is suitable for the targeted application, that uses an interconnection network similar to that found in FPGAs. While the proposed model can be extended for real PCB routers, it provides the basis and foundation towards a netlist generator for general PCB routers.

Acknowledgements

The authors would like to thank NSERC and Gestion TechnoCap Inc. for financial support. The authors want to mention OpenMoko for their willingness in releasing publicly parts of their design, HyperChip for providing one of their industrial-quality PCB project, and Gaétan Décarie for his support, and for sharing experience and for providing several PCB netlists.

Chapitre 6

Discussion générale

Les contributions présentées ci-avant découlent du contexte des recherches, des contraintes du projet, des idées qui sortent de la littérature aujourd'hui. Une large part des contributions sont applicables à divers algorithmes de routage dans d'autres domaines, cependant leur intérêt mérite d'être discuté.

Avant toutes choses, la première contrainte importante des recherches effectuées proviennent du réseau d'interconnexions. Celui-ci partage des similitudes fortes avec ceux embarqués sur *FPGA*, mais conservent des spécificités qui rend difficile, voire impossible, toute comparaison directe des résultats de routage obtenus. Ce réseau a été bâti pour répondre à certains critères, notamment la surface occupée, la densité des interconnexions, la longueur maximum des liens, la tolérance aux pannes, l'intégration à l'échelle de la tranche. Au cœur du réseau se situe le *crossbar*, qui a été étudié principalement par l'auteur de cette thèse. Les résultats de cette étude ont été publiés à la conférence *NEWCAS* 2008 [15]. Ce réseau a été ensuite testé sur une puce fabriquée spécialement pour une version miniature du *WaferIC* complet. Cette version nommée *TestChip* V1.0 comportait une matrice de 3×3 cellules, et les liens entre cellules de courtes distances (1 et 2) ont pu être testés et validés. C'est par ce moyen, ainsi que par simulation après placement que les délais d'interconnexions ont été extraits. Ces résultats ont été publiés à la conférence *MNRC* 2009 [14] sous forme d'article, et a obtenu le deuxième prix au « *Best Paper Award* » de la conférence.

Du côté des contraintes de temps de calcul, d'occupation mémoire et d'implémentation, une analyse détaillée du flot de travail du *WaferBoard* a été réalisée avant les recherches spécifiques au routage. Ainsi, il a été démontré de l'importance de temps de calcul restreints ; également, les interactions du routeur avec les autres outils logiciels pour le WaferBoard ont été déterminés. Ces travaux ont été publiés à la conférence *ISCAS* 2008 [16]. Ces travaux préliminaires ont permis de bien comprendre le contexte du projet, les contraintes uniques qui s'y appliquent ainsi que les points communs avec d'autres technologies. Les choix algorithmiques et architecturaux prennent racines dans les recherches qui ont mené à la publication de ces trois articles de conférences.

En ce qui concerne directement l'algorithme de routage, une des contributions majeure est le routage par l'aléatoire. Il sera montré dans la revue de littérature du chapitre 2 que ce type d'approche n'a pas été étudié. Or, il est démontré dans cette thèse que l'accélération du routage peut atteindre plus d'un ordre de grandeur : ce résultat majeur étonne, et

remet sans doute en question sa crédibilité. Par exemple, serait-il possible d'utiliser une telle approche et ainsi diviser par 10 les temps de routage sur *FPGA* ? Il s'avère que reproduire les résultats obtenus sur un circuit intégré à l'échelle de la tranche, auprès d'un circuit de quelques centaines de millimètres carrés n'est pas aussi simple. Premièrement, les techniques de routage aléatoire proposées ne sont pas totalement aléatoires : elles sont guidées par construction, c'est à dire que l'espace de recherche est défini. Cet espace donne aux solutions potentielles générées une garantie d'optimalité, et un taux de succès élevé, ce qui donne d'autant plus d'efficacité à une telle approche. Il est tout à fait possible d'utiliser une telle méthodologie auprès d'autres technologies.

Cependant, un certain nombre de spécificités du *WaferIC* n'autoriseront probablement pas de gains aussi importants sur *PCB*, *VLSI*, *FPGA*. Toutes ces technologies utilisent un algorithme de placement intelligent (bien que sur *PCB* il s'agisse souvent d'un ingénieur). Ce placement permet de très largement réduire les pics de la demande en ressources de routage. Un étalement intelligent des composants permet donc de réduire les ressources de routages nécessaires. Or, ce placement n'existe pas pour le *WaferBoard* : pour compenser, une forte densité d'interconnexions est mis à la disposition du routage, pour faire face à un maximum de conditions de placement. À cela s'ajoute les problèmes de tolérance aux pannes. Du fait de l'intégration à l'échelle de la tranche, le réseau d'interconnexions doit autoriser des routages importants, quand bien même une partie des ressources sur une zone soit non-fonctionnelle après fabrication. Le réseau actuel ne permet bien entendu pas de circonvenir à toutes les situations imaginables, cependant l'objectif était lors des phases de conception du matériel d'utiliser tout l'espace disponible en ressources supplémentaires. Les taux de défectuosités ne sont pas connus précisément, et les utilisations du WaferBoard étant très variés, le réseau est fortement connecté. Les approches aléatoires sont rapides lorsque la probabilité de générer une solution viable est importante. Or, si localement la densité des demande en ressources est par endroit importante sur le *WaferIC*, globalement elle est plus faible que sur *FPGA* ou *VLSI*. Cet aspect aide énormément au succès des approches aléatoires appliquées au *WaferIC*. L'effet de la distribution et du taux de défectuosités sur les performances du routeur n'a pas encore été étudié.

Comparativement aux bancs d'essais pour *VLSI* ou *FPGA*, le routage pour *WaferIC* semble donc facile, car le rapport entre la demande en ressources et la disponibilité des ressources est faible. Cependant, deux contraintes spécifiques contre-balancent ceci. La première est la forte connectivité du réseau, cumulée aux distances importantes parcourues par les *net*. Les routeurs en labyrinthe performants, tels le A^*, ont une complexité fonction du nombre de nœuds et d'arcs. Le nombre important d'arcs au sein du *WaferIC* ralentit le A^* dans les zones congestionnées. En effet, sur le réseau du *WaferIC*, le nombre de routes possibles est exponentiel avec la distance à parcourir. Deuxièmement, le temps de calculs acceptable pour le *WaferIC* est très petit, de l'ordre de la minute. Sur *FPGA*, *VLSI* ou *PCB*, les recherches se sont au contraire focalisées sur le routage dans des conditions difficiles (congestions particulièrement importantes) et sur l'amélioration de la qualité des résultats, plutôt que sur la diminution des temps de calcul, moins prioritaires.

De manière générale, les algorithmes proposés dans cette thèse sont applicables à d'autres technologies. L'impact des techniques de routage aléatoire est très certainement plus faible que sur le *WaferIC*, mais méritent d'être étudiées,

notamment sur *FPGA*. Par contre, les recherches concernant le modèle de *netlist* pour *PCB* embrassent une problématique plus large que le *WaferBoard* lui-même, et constituent une brique intéressante pour la recherche, et probablement la contribution la plus importante de cette thèse. Cette section de la thèse (chapitre 5) a mené à la découverte de limitations de la loi de Rent sur *PCB*. Historiquement, la loi de Rent à été étudiée sur *PCB* ; cette thèse démontre qu'aujourd'hui cette loi n'est plus applicable. Les raisons de ce changement sont très probablement dû à une forte intégration des composants électroniques. En 1970 lors des premières études et découverte de cette loi, chaque composant avait un rôle spécifique et des centaines étaient régulièrement posés sur un *PCB*. Aujourd'hui quelques composants majeurs seulement sont placés, et beaucoup de composants très petits et passifs sont utilisés en complément, détruisant une des hypothèse de la loi de Rent.

Bien que la loi soit encore aujourd'hui valide sur FPGA et VLSI, il est possible avec l'explosion de l'utilisation d'IP au sein de puces électronique et leur intégration continue, que dans un futur proche le même phénomène se produise pour VLSI et/ou FPGA. Il est encore trop tôt pour en être certain, mais si cela s'avère vrai, les techniques de modélisations proposées dans cette thèse prendraient une valeur importante dans les domaines des FPGA et des routeurs VLSI.

Conclusion

Cette thèse a présenté les recherches effectuées pour la réalisation d'un algorithme de routage du *WaferIC*. Cette technologie se compare sur certains points aux technologies connues tels les *FPGA*, *PCB* ou encore les contraintes rencontrées par les outils *VLSI*. Ainsi, la taille du réseau d'interconnexions est comparable à celle des routeurs pour *VLSI*, et ce réseau partage certains aspects des *FPGA* (longueurs disponibles variées, densité des interconnexions). D'autres aspects importants sont au contraire uniques, notamment les contraintes de temps de calcul, de l'ordre de la minute, l'architecture du réseau d'interconnexions, ou encore l'intersection des contraintes du routage sur des *netlist PCB* (taille, densité) appliquées à un réseau sur circuit intégré.

Au cours de ces recherches, les contributions suivantes ont été apportées :

- Un modèle de *netlist* pour *PCB* ;
- Un algorithme de génération basée sur ce modèle ;
- Une méthode de calcul d'un chemin de plus petit délai dans le graphe intégré au *WaferIC* ;
- Une preuve mathématique que cette méthode trouve toujours le chemin de plus petit délai, dont la complexité algorithmique est linéaire avec le nombre de noeuds de la route finale ;
- Une méthode de résolution des conflits qui utilise l'aléatoire, nommée routage par permutations. Les accélérations obtenues sont comprises entre un facteur de 6 et de 14 pour les *netlist* courants attendus pour l'application ;
- Une parallélisation du routage par permutations, qui réduit les temps de calcul entre 14 % et 529 % sur une machine avec 4 processeurs, pour une complexité d'implémentation faible ;
- La réutilisation des résultats de permutations pour réduire le temps de résolution des conflits restants par un routeur labyrinthe, par le routage des seules zones en conflits Ceci apporte un accélération d'un facteur qui varie de 2 à 80 sur le *WaferIC*, pour une dégradation de la qualité du routage faible pour l'application ;
- Sous la forme d'une extension non-triviale de l'algorithme *RCV*, un algorithme d'allocation dynamique des délais pour l'équilibrage de ceux-ci dans des groupes de taille arbitraire ;
- Devant les temps de routage excessifs lors de l'équilibrage de délais malgré les importantes contributions précédentes, un algorithme de routage plus rapide est proposé. Il fonctionne à partir d'une table d'équivalences de délais, qui offre une accélération comprise entre 71 % et 180 % pour un équilibrage raisonnable sans permutations, et entre 86 % et 213 % avec permutations ;

– Une modification de la fonction de coût utilisée par *RCV*, et qui offre une accélération d'un facteur entre 10 et 180 pour les *netlist* attendus, permettant finalement de router un *netlist* de densité très important (30 %) avec un équilibrage raisonnable en 8 minutes.

Le modèle de *netlist* pour *PCB* constitue une contribution importante. En effet, un tel modèle, adapté aux *PCB* modernes, n'avait encore jamais été publié, bien que des modèles pour *FPGA* ou *VLSI* existent. Au cours de la construction de ce modèle, il a également été découvert que la loi de Rent, largement utilisée sur d'autres technologies, ne pouvait s'appliquer aux *PCB* modernes. En effet, la forte intégration des composants électroniques adjointe aux nombreux choix de boîtiers pour circuits confère aux *PCB* une faible régularité spatiale. Cette découverte a mené à la construction d'un modèle original, simple et robuste de *netlist* pour *PCB*. Un algorithme de génération de *netlist* en a été déduit. Cet algorithme utilise une approche adaptée à la rareté des *netlist PCB* réelles ; là où les algorithmes générateurs de *netlist* pour *FPGA* ou *VLSI* utilisent aujourd'hui des *netlist* réelles pour en construire des clones, l'algorithme proposé utilise une approche stochastique qui ne nécessite pas de *netlist* source à chaque génération.

Le modèle de *netlist* a servi ensuite à générer des *netlist* adaptées au routeur pour le *WaferIC*. L'algorithme de routage réalisé au cours de ces recherches est aujourd'hui intégré à la suite de logiciel du *WaferBoard*, pleinement fonctionnel. Il utilise en son cœur un A* combiné à un ensemble de techniques novatrices pour en accélérer le calcul, et qui forment un autre ensemble de contributions. En effet, l'utilisation d'un A* seul ne permet pas d'atteindre les temps de calcul désirés, en particulier lors de l'équilibrage de délais. De plus, l'architecture du réseau nécessite la construction d'une heuristique admissible [1] pour utiliser un A* nettement plus rapide que Dijkstra. Une méthode de calcul du plus court chemin dans un réseau multi-dimensionnel a donc été proposée en premier lieu. De plus, une preuve mathématique démontre que cette méthode trouve toujours le plus court chemin dans un tel réseau. Il a été également démontré que la complexité algorithmique de cette approche dépend seulement du nombre de noeuds de la route construite, indépendamment de la taille du graphe. Il s'agit d'un résultat important, car les routeurs de type labyrinthe, comme le A*, ont une complexité qui dépend du nombre de noeuds et d'arcs du graphe. Ainsi, l'approche peut servir d'une part à effectuer la première passe de routage (chaque *net* indépendamment des autres) très rapidement comparativement au A*, et d'autres part d'heuristique de prédiction admissible de délai d'une route pour le A*. En deuxième lieu, et pour accélérer les temps de calculs et atteindre les objectifs fixés par les besoins de l'application, un algorithme de résolution de conflit a été développé. Cette approche consiste à générer un grand nombre de solutions optimales en terme de délai, et de vérifier si une est valide. Simple, cette approche réduit considérablement le nombre de routes à calculer par le A* et accélère jusqu'à 100 fois les temps de calcul. Elle peut être parallélisée pour offrir un gain appréciable d'environ 10 % sur 4 cœurs pour des *netlist* denses, et d'un facteur proche du nombre de *CPU* disponibles pour des *netlist* de faible densité (5-10 %). Cette technique est utilisable par des algorithmes de routage sur *PCB*, *VLSI* ou *FPGA*, bien que les gains espérés sur ces technologies soient plus modérés. Cette contribution met

1. L'heuristique ne surestime jamais le coût d'une route à priori.

en lumière l'intérêt du routage par une méthode aléatoire : il n'est pas envisageable de l'utiliser seule, mais il s'agit d'un complément qui permet de réduire les temps de calcul, sans pour autant impacter la qualité de la solution finale. Contribution supplémentaire, cette approche peut servir à conserver la solution la moins conflictuelle. Ensuite, celle-ci peut être utilisée pour trouver une solution non-conflictuelle plus rapidement. En effet, il a été proposé de router avec le A^* non pas du point de départ au point d'arrivée, mais seulement entre chaque segment conflictuel. Les résutlats montrent que cette approche impacte faiblement la qualité du routage, tout en proposant une accélération jusqu'à un facteur 11 pour le *WaferIC*.

Le routage seul n'est pas suffisant pour l'application du *WaferBoard* : du fait des délais importants du réseau d'interconnexions comparé aux traces de cuivre d'un *PCB*, il est critique pour le succès du projet de supporter les contraintes d'équilibrage de délai. Bien que des algorithmes pour *PCB* et *VLSI* aient été publiés, le réseau d'interconnexions est très différent sur le *WaferIC*, et plus près des *FPGA*. Cependant, il n'y a pas de publication dans ce domaine pour *FPGA* : l'algorithme *RCV* a servi de base pour l'équilibrage des délais sur le *WaferIC*, et a été étendu par plusieurs contributions. D'abord, une technique d'allocation dynamique des délais a été proposée, pour permettre l'équilibrage au sein de groupes de taille arbitraire. Cette technique relaxe les bornes de délai minimum et maximum de chaque *net* d'un groupe lorsque l'un des membre ne peut être routé dans les bornes spécifiées. De plus, deux contributions pour l'accélération des temps de calcul ont été présentées. Une technique de routage par table d'équivalences a été proposée, qui permet de limiter l'impact sur le temps de calcul de l'équilibrage, d'un facteur supérieur à 4. Cette technique possède des similarités à une autre publiée très récemment (mars 2011) sur *PCB*, bien que les différences d'application rende les deux approches significativement différentes. Deuxièmement, une modification de la fonction de coût de *RCV* a été proposée, et cette contribution apporte un facteur d'accélération très important, pouvant atteindre 400 sur l'application visée.

Ces recherches ont permis toutes ces contributions, et d'autres recherches devraient être poursuivies dans la continuité. Deux axes de recherche principaux se détachent. Le premier est orienté matériel, soit l'optimisation de la surface occupée par le réseau d'interconnexions et de l'architecture du réseau qui a une influence directe sur la taille physique de la cellule. Le deuxième axe de recherches se concentrerait sur l'algorithme de routage proprement dit, et notamment certaines fonctionnalités qu'il serait intéressant d'apporter.

Le réseau d'interconnexions occupe une surface très importante au sein de la cellule du *WaferIC*. Une nouvelle architecture permettrait sans doute de limiter la surface, sans forcément limiter de manière importante les capacités de routage et de tolérance aux défectuosités. De plus, il faudrait réduire les délais des longues interconnexions dûes au *crossbar*. Pour ce faire, une analyse de l'adéquation entre le réseau actuel et son utilisation (routage) est nécessaire. Environ un quart de la surface de la cellule est dédiée au *crossbar*, ce qui en fait le premier module numérique en terme de surface. De nombreux nouveaux modules analogiques doivent être ajoutés (senseurs de température, bus analogique, ...) et si quelques modules prendront place au sein de zones non occupées dans certaines cellules, la

réduction de la surface du *crossbar* offrirait des opportunités pour de nouvelles fonctionnalités. Optimiser le réseau d'interconnexions est possible, avec d'un côté une analyse de surface réalisable par synthèse des éléments simples en première approximation, et de l'autre la modification de l'algorithme de routage pour tenir compte des architectures explorées, afin d'en déterminer la routabilité.

Pour compléter cette analyse, un détail de l'architecture du réseau devrait être revu : si chaque cellule supporte deux billes, c'est parce que le *crossbar* possède deux entrées et deux sorties dédiées. Or, ce nombre combiné à la taille de la cellule détermine les technologies de boîtiers de composants supportés par le *WaferIC*. Il est proposé de réaliser une étude de la surface optimale d'une cellule, fonction de la probabilité de défaut des différents modules de la cellule, de la position physique de celle-ci sur le *WaferIC*, des architectures du réseau explorées et de leur routabilité. Il est probable que des gains importants en terme de robustesse (taux de panne) et de surface libre par cellule (donc de fonctionnalités embarquées) sont possibles, et pourraient découler d'une telle étude.

En ce qui concerne le deuxième axe de recherche, l'algorithme de routage proprement dit, plusieurs fonctionnalités méritent d'être ajoutées. En premier lieu, il faudrait ajouter à l'équilibrage des délais d'un groupe l'équilibrage d'un *net*. En effet, certains *net* avec un fanout élevé, tel une horloge, nécessitent souvent un équilibrage des branches. Cette fonctionnalité n'a pas été implémentée, car non-prioritaire et moins porteuse sur le plan de la recherche. En effet, de nombreux algorithmes existent pour assigner les contraintes de délai à chaque nœud : Zero-Slack Allocation[99], ou encore Iterative-Minimax-PERT[59] sont des algorithmes capables de réaliser cette allocation. Ensuite, le A^* développé pour les recherches présentées ici est capable de router en tenant compte de ces contraintes. Les recherches sur l'équilibrage des délais ont établi une faiblesse du WaferBoard : l'absence d'algorithme de placement. Or, aussi intelligent puisse-t-il être, aucun routeur ne peut faire un routage de bonne qualité si le placement est mal réalisé. Cependant, une piste de recherche s'ouvre ici : à partir des résultats du routage, il est possible d'extraire les chemins critiques qui limitent les performances du circuit. À partir de ces informations, il faudrait construire un algorithme qui recherche un placement plus optimal pour le proposer à l'utilisateur.

De plus, le routeur actuel utilise l'algorithme *FLUTE* pour effectuer les calculs des points Steiner [2]. Il s'agit aujourd'hui de l'algorithme publié le plus performant, mais il considère que le coût d'un arbre est directement proportionnelle à la distance parcourue. Or, ce n'est pas le cas dans le réseau du *WaferIC* : la fonction du délai (c'est-à-dire le coût du point de vue routage) n'est pas directement proportionnelle à la distance parcourue : la fonction n'est pas monotone. En conséquence, les solutions générées par *FLUTE* sont sous-optimales en terme de délai, et un gain, certes limité, pourrait être obtenu ici. Le gain est limité car par définition, la proportion des *net* qui sont travaillés par *FLUTE* est faible. En effet, le calcul des points Steiner ne s'effectue que sur des *net* de fanout supérieur à 1, et il devient en général de plus en plus important avec l'augmentation du fanout. Cependant, la majeure partie des *net* sont de fanout 1 sur un *PCB*, et d'autant plus sur le *WaferIC* que les alimentations ne sont pas routées - ces *net* ne sont donc pas impactés par

2. Étant donné un arbre (ici, un *net* dont le fanout est supérieur à 1), trouver l'arbre dont la somme des distances est minimal (il est autorisé d'insérer des points appelés points Steiner).

FLUTE. Des recherches sur cet axe pourraient néanmoins se révéler plus importantes en fonction des optimisations effectuées sur le réseau d'interconnexions, qui réduiraient probablement les capacités de routage.

Les axes de recherches évoqués ici sont probablement la pointe émergée de l'iceberg : car c'est en avançant dans cette thèse que de nouvelles idées se sont révélées, et de nouveaux champs de recherche apparus.

Bibliographie

[1] A. Juttner, B. Szviatovszki, I. Mecs, and Z. Rajko, "Lagrange relaxation based method for the QoS routing problem," 2001.

[2] Y. Xiao, K. Thulasiraman, and G. Xue, "GEN-LARAC : a generalized approach to the constrained shortest path problem under multiple additive constraints," in *Algorithms and Computation*, pp. 92–105, 2005.

[3] G. Y. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, vol. 10, no. 4, pp. 293–309, 1980.

[4] D. Yang and S. A. Zenios, "A scalable parallel interior point algorithm for stochastic linear programmingand robust optimization," *Comput. Optim. Appl.*, vol. 7, no. 1, pp. 143–158, 1997.

[5] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *Siam Symposium on discrete algorithms*, pp. 166—173, 2001.

[6] H. Crowder, E. L. Johnson, and M. Padberg, "Solving Large-Scale Zero-One linear programming problems," *Operations Research*, vol. 31, pp. 803–834, Oct. 1983.

[7] L. Behjat and A. Chiang, "Fast integer linear programming based models for VLSI global routing," in *IEEE International Symposium on Circuits and Systems (ISCAS), 23-26 May 2005*, vol. Vol. 6 of *IEEE International Symposium on Circuits and Systems (ISCAS) (IEEE Cat. No. 05CH37618)*, (Piscataway, NJ, USA), pp. 6238–43, IEEE, 2005. Copyright 2005, IEE.

[8] C. Pornavalai, G. Chakraborty, and N. Shiratori, "Routing with multiple QoS requirements for supporting multimedia applications," *Telecommunication Systems*, vol. 9, no. 3, pp. 357–373, 1998.

[9] J. Pistorius and M. Hutton, "Placement rent exponent calculation methods, temporal behaviour and FPGA architecture evaluation," in *Proceedings of the 2003 international workshop on System-level interconnect prediction*, SLIP '03, (New York, NY, USA), pp. 31–38, ACM, 2003.

[10] A. M. Mohsen, "Field programmable printed circuit board, US5377124(A)," Dec. 1994.

[11] V. E. Benes, *Mathematical theory of connecting networks and telephone traffic*. Academic Press, 1965.

[12] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture : Survey and challenges," *Found. Trends Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, 2008.

[13] ISPD, "Ispd 2008 global routing contest," 2008.

[14] E. Lepercq, O. Valorge, Y. Basile-Bellavance, N. Laflamme-Mayer, Y. Blaquiere, and Y. Savaria, "An interconnection network for a novel reconfigurable circuit board," in *Microsystems and Nanoelectronics Research Conference, 2009. MNRC 2009. 2nd*, pp. 53–56, 2009.

[15] R. Norman, E. Lepercq, Y. Blaquiere, O. Valorge, Y. Basile-Bellavance, R. Prytula, and Y. Savaria, "An interconnection network for a novel reconfigurable circuit board," in *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, pp. 129–132, 2008.

[16] E. Lepercq, Y. Blaquiere, R. Norman, and Y. Savaria, "Workflow for an electronic configurable prototyping system," in *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pp. 2005 –2008, May 2009.

[17] A. Schrijver, "On the history of combinatorial optimization (till 1960)," 1960.

[18] S. E. Dreyfus, "An appraisal of some Shortest-Path algorithms," *Operations Research*, vol. 17, pp. 395–412, Jan. 1969.

[19] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec. 1959.

[20] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 107, 100, 1968.

[21] R. Prim, "Shortest connection networks and some generalizations," *Bell System Technology Journal*, vol. 36, pp. 1401, 1389, 1957.

[22] J. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, pp. 50, 48, Feb. 1956.

[23] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, 1960.

[24] G. Dantzig, *Origin of the Simplex Method.* 1947.

[25] A. M. Geoffrion, "Lagrangian relaxation for integer programming," in *50 Years of Integer Programming 1958-2008*, pp. 243–281, 1974.

[26] R. Bellman, "Dynamic programming and lagrange multipliers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 42, no. 10, p. 767, 1956.

[27] J. Lee, Y. Won, S. Sahni, and E. Shragowitz, "Parallel algorithms for physical design," in *Circuits and Systems, 1988., IEEE International Symposium on*, pp. 325 –328 vol.1, June 1988.

[28] L. Qing-hua, L. Ken-li, W. Duo-qiang, and X. Jia-jung, "A scalable parallel algorithm of linear programming," *Mini-Micro Systems*, vol. 24, pp. 1718–21, Sept. 2003. Copyright 2004, IEE.

[29] H. Sun and S. Aretbi, "Global routing for VLSI standard cells," in *Electrical and Computer Engineering, 2004. Canadian Conference on*, vol. 1, pp. 485 – 488 Vol.1, May 2004.

[30] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[31] V. Cerny, "Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 51, 41, 1985.

[32] C. Lee, "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 2, pp. 364–365, 1961.

[33] W. E. Nak, K. Taewhan, and M. K. Chong, "A router for symmetrical FPGAs based on exact routing density evaluation," 2001.

[34] R. Fung, V. Betz, and W. Chow, "Simultaneous short-path and long-path timing optimization for FPGAs," in *Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pp. 838–845, IEEE Computer Society, 2004.

[35] M. Saeedi, M. S. Zamani, and A. Jahanian, "Congestion prediction : from metric definition to routing estimation," in *Microelectronics, 2005. ICM 2005. The 17th International Conference on*, pp. 183 – 188, 2005.

[36] M. Saeedi, M. Zamani, and A. Jahanian, "Evaluation, prediction and reduction of routing congestion," *Microelectronics Journal*, vol. 38, no. 8-9, pp. 942–58, 2007. Copyright 2007, The Institution of Engineering and Technology.

[37] Z. Li, C. Alpert, S. Quay, S. Sapatnekar, and W. Shi, "Probabilistic congestion prediction with partial blockages," in *2007 IEEE International Symposium on Quality of Electronic Design, 26-28 March 2007*, (Los Alamitos, CA, USA), p. 6 pp., IEEE Computer Society, 2007. Copyright 2007, The Institution of Engineering and Technology.

[38] R. Fung, V. Betz, and W. Chow, "Slack allocation and routing to improve FPGA timing while repairing Short-Path violations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 686–697, 2008.

[39] M. Ozdal and M. Wong, "Archer : a history-driven global routing algorithm," in *2007 IEEE/ACM International Conference on Computer Aided Design, 4-8 Nov. 2007*, 2007 IEEE/ACM International Conference on Computer Aided Design, (Piscataway, NJ, USA), pp. 488–95, IEEE, 2007.

[40] T. Ho, "PIXAR : a performance-driven x-architecture router based on a novel multilevel framework," *Integration, the VLSI Journal*, vol. 42, pp. 400–408, June 2009.

[41] Y. Xu, Y. Zhang, and C. Chu, "FastRoute 4.0 : global router with efficient via minimization," in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, (Yokohama, Japan), pp. 576–581, IEEE Press, 2009.

[42] T. Wu, A. Davoodi, and J. T. Linderoth, "GRIP : scalable 3D global routing using integer programming," in *Proceedings of the 46th Annual Design Automation Conference*, (San Francisco, California), pp. 320–325, ACM, 2009.

[43] D. Blokh and G. Gutin, "An approximation algorithm for combinatorial optimization problems with two parameters," tech. rep., University of Southern Denmark, 1995.

[44] W. Lee, M. Hluchyi, and P. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *Network, IEEE*, pp. 46–55, 1995.

[45] Z. Wang and J. Crowcroft, "Quality-of-Service routing for supporting multimedia applications," *IEEE Journal of Selected Areas in Communications*, vol. 14, no. 7, pp. 1234, 1228, 1996.

[46] Z. Jia and P. Varaiya, "Heuristic methods for delay constrained least cost routing using k-shortest-paths," *Institute of Electrical and Electronics Engineers. Transactions on Automatic Control*, vol. 51, no. 4, pp. 707–712, 2006.

[47] Z. Jia and P. Varaiya, "Heuristic methods for delay constrained least cost routing using k-shortest-paths," *INFOCOM*, 2001.

[48] K. Zhang, H. Zhang, and J. Xu, "An efficient distributed dynamic multicast routing with delay and delay variation constraints," in *Networking and Mobile Computing*, pp. 198–207, 2005.

[49] W. Ben-Ameur and A. Ouorou, "Mathematical models of the delay constrained routing problem," *Algorithmic Operations Research*, vol. 1, no. 2, pp. 94–103, 2006.

[50] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," *ACM Trans. Algorithms*, vol. 4, no. 1, pp. 1–17, 2008.

[51] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM J. Discret. Math.*, vol. 13, no. 4, pp. 505–520, 2000.

[52] M. Balinsky and P. Wolfe, "Nondifferentiable optimization," *Math. Programming Stud.*, 1975.

[53] Y. Xiao, K. Thulasiraman, and G. Xue, "Constrained shortest Link-Disjoint paths selection : A network programming based approach.," *IEEE Transactions on Circuits & Systems Part I*, vol. 53, pp. 1174–1187, May 2006.

[54] ISPD, "Ispd 2007 global routing contest," 2007.

[55] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, (San Jose, California), pp. 496–502, IEEE Press, 2007.

[56] H. Bolfinger, "A mature DA system for PC layout," pp. 85–99, 1979.

[57] J. Soukup, "Global router," in *Proceedings of the 16th Design Automation Conference*, (San Diego, CA, United States), pp. 481–484, IEEE Press, 1979.

[58] P. S. Hauge, R. Nair, and E. J. Yoffa, "Circuit placement for predictable performance," in *IEEE International Conference on Computer-Aided Design : ICCAD-87 - Digest of Technical Papers.*, (Santa Clara, CA, USA), pp. 88–91, IEEE, 1987. Compendex.

[59] J. Frankle, "Iterative and adaptive slack allocation for performance-driven layout and FPGA routing," in *Proceedings of the 29th ACM/IEEE Design Automation Conference*, (Anaheim, California, United States), pp. 536–542, IEEE Computer Society Press, 1992.

[60] G. Wu, J. Lin, and Y. Chang, "Performance-driven placement for dynamically reconfigurable FPGAs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 7, no. 4, pp. 628–642, 2002.

[61] Y. Chang, Y. Lee, and T. Wang, "NTHU-Route 2.0 : a fast and stable global router," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, (San Jose, California), pp. 338–343, IEEE Press, 2008.

[62] V. Betz and J. Rose, "VPR : a new packing, placement and routing tool for FPGA research," *Field-Programmable Logic and Applications*, pp. 213—222, 1997.

[63] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Springer, 1st ed., Mar. 1999.

[64] L. McMurchie and C. Ebeling, "PathFinder : a Negotiation-Based Performance-Driven router for FPGAs," in *Field-Programmable Gate Arrays, 1995. FPGA '95. Proceedings of the Third International ACM Symposium on*, pp. 111–117, 1995.

[65] J. S. Swartz, V. Betz, and J. Rose, "A fast routability-driven router for FPGAs," *6th international workshop on field-programmable gate arrays*, pp. 140—149, 1998.

[66] K. Zhu, Y.-w. Chang, and D. F. Wong, "Timing-Driven routing for Symmetrical-Array-Based FPGAs," *Trans. on design automation of electronic systems*, vol. 5, pp. 433—450, 2000.

[67] L. Scheffer, L. Lavagno, and G. E. Martin, *EDA for IC implementation, circuit design, and process technology*. CRC Press, Mar. 2006.

[68] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs*. Boston, MA : Springer US, 2009.

[69] M. M. Ozdal, "A provably good algorithm for high performance bus routing," *In proc. of IEEE/ACM intl. conf. on computer-aided design (ICCAD)*, 2004.

[70] M. M. Ozdal, *Routing algorithms for high-performance VLSI packaging*. PhD thesis, University of Illinois at Urbana-Champaign, 2005.

[71] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, Jan. 1979.

[72] R. Widyono, "The design and evaluation of routing algorithms for Real-Time channels," *International Computer Science Institute*, 1994.

[73] C. Pornavalai, G. Chakraborty, and N. Shiratori, "QoS based routing algorithm in integrated services packet networks," in *Proceedings of the 1997 International Conference on Network Protocols (ICNP '97)*, p. 167, IEEE Computer Society, 1997.

[74] M. M. Ozdal and M. D. F. Wong, "Length-Matching routing for High-Speed printed circuit boards," in *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, p. 394, IEEE Computer Society, 2003.

[75] C. Hsu, "General river routing algorithm," in *Proceedings of the 20th Design Automation Conference*, (Miami Beach, Florida, United States), pp. 578–583, IEEE Press, 1983.

[76] M. M. Ozdal and M. D. F. Wong, "A Length-Matching routing algorithm for High-Performance printed circuit boards," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 12, pp. 2784 –2794, 2006.

[77] M. M. Ozdal and R. F. Hentschke, "Exact route matching algorithms for analog and mixed signal integrated circuits," in *Proceedings of the 2009 International Conference on Computer-Aided Design*, (San Jose, California), pp. 231–238, ACM, 2009.

[78] E. Amouri, H. Mrabet, Z. Marrakchi, and H. Mehrez, "Placement and routing techniques to improve delay balance of WDDL netlist in MFPGA," in *2009 16th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2009, December 13, 2009 - December 16, 2009*, 2009 16th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2009, (Yasmine Hammamet, Tunisia), pp. 791–794, IEEE Computer Society, 2009. Compendex.

[79] Y. Kubo, H. Miyashita, Y. Kajitani, and K. Tateishi, "Equidistance routing in high-speed VLSI layout design," *Integration, the VLSI Journal*, vol. 38, no. 3, pp. 439–449, 2005.

[80] T. Yan and M. D. F. Wong, "BSG-Route : a length-matching router for general topology," in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, (San Jose, California), pp. 499–505, IEEE Press, 2008.

[81] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module packing based on the BSG-structure and IC layout applications," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 6, pp. 519 –530, 1998.

[82] J. Yan and Z. Chen, "Obstacle-aware length-matching bus routing," in *Proceedings of the 2011 international symposium on Physical design*, ISPD '11, (New York, NY, USA), pp. 61–68, ACM, 2011. ACM ID : 1960412.

[83] IBM, "Ibm-place 2.0 benchmark suits," 2002.

[84] F. Brglez, “Acm/sigda benchmarks electronic newsletter dac’93 edition,” 1993.

[85] N. Abboud, M. Grotschel, and T. Koch, “Mathematical methods for physical layout of printed circuit boards : an overview,” *OR Spectrum*, vol. 30, pp. 453–468, June 2008.

[86] D. Stroobandt, P. Verplaetse, and J. van Campenhout, “Generating synthetic benchmark circuits for evaluating CAD tools,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 9, pp. 1011–1022, 2000.

[87] B. Landman and R. Russo, “On a pin versus block relationship for partitions of logic graphs,” *Computers, IEEE Transactions on*, vol. C-20, no. 12, pp. 1469–1479, 1971.

[88] D. Stroobandt, “On an efficient method for estimating the interconnection complexity of designs and on the existence of region III in rent’s rule,” in *Great Lakes Symposium on VLSI*, (Los Alamitos, CA, USA), p. 330, IEEE Computer Society, 1999.

[89] W. E. Donath, “Wire length distribution for placements of computer logic,” 1981.

[90] D. Stroobandt, *A priori Wire Length Estimates for Digital Design*. Springer, 1 ed., Mar. 2001.

[91] D. Stroobandt and F. J. Kurdahi, “On the characterization of multi-point nets in electronic designs,” *Proceedings of the 8th Great Lakes Symposium on VLSI*, pp. 344–350, 1998.

[92] T. Wan and M. Chrzanowska-Jeske, “Prediction of interconnect net-degree distribution based on rent’s rule,” in *Proceedings of the 2004 international workshop on System level interconnect prediction*, pp. 107–114, ACM, Feb. 2004.

[93] D. Stroobandt and H. Van Marck, “Efficient representation of interconnection length distributions using generating polynomials,” in *Proceedings of the 2000 international workshop on System-level interconnect prediction*, pp. 99–105, ACM, 2000.

[94] M. Hutton, J. Rose, and D. Corneil, “Automatic generation of synthetic sequential benchmark circuits,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 21, no. 8, pp. 928–940, 2002.

[95] M. R. Garey and D. S. Johnson, “The rectilinear steiner tree problem is NP-Complete,” *SIAM Journal on Applied Mathematics*, vol. 32, pp. 826–834, June 1977. ArticleType : primary_article / Full publication date : Jun., 1977 / Copyright Â© 1977 Society for Industrial and Applied Mathematics.

[96] R. Norman, O. Valorge, Y. Blaquiere, E. Lepercq, Y. Basile-Bellavance, Y. El-Alaoui, R. Prytula, and Y. Savaria, “An active reconfigurable circuit board,” in *Circuits and Systems and TAISA Conference, 2008. Joint 6th International IEEE Northeast Workshop on*, pp. 351–354, 2008.

[97] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, “Architecture of field-programmable gate arrays,” *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1013–1029, 1993.

[98] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar, “A comparative study of two boolean formulations of FPGA detailed routing constraints,” *Computers, IEEE Transactions on*, vol. 53, no. 6, pp. 688–696, 2004.

[99] R. Nair, “A simple yet effective technique for global wiring,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 6, no. 2, pp. 165–172, 1987.

[100] P. Hart, N. Nilsson, and B. Raphael, “Correction to "A formal basis for the heuristic determination of minimum cost paths",” *SIGART Bull.*, no. 37, pp. 29, 28, 1972.

[101] M. Cho, K. Lu, K. Yuan, and D. Z. Pan, "BoxRouter 2.0 : A hybrid and robust global router with layer assignment for routability," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, pp. 1–21, 2009.

[102] C. Chu and Y. Wong, "Fast and accurate rectilinear steiner minimal tree algorithm for VLSI design," in *Proceedings of the 2005 international symposium on Physical design*, (San Francisco, California, USA), pp. 28–35, ACM, 2005.

[103] J. Held, J. Bautista, and S. Koehl, "From a few cores to many : A tera-scale computing research overview," white paper, Intel Corporation, 2006.

[104] E. W. Dijkstra, *A Discipline of Programming*. Prentice Hall, Inc., Oct. 1976.

[105] M. Pan and C. Chu, "FastRoute : a step to integrate global routing into placement," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, (San Jose, California), pp. 464–471, ACM, 2006.

[106] C. Chris and Y. Wong, "Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design," *IEEE transactions on computer-aided design*, 2004.

[107] Xilinx, "Xilinx timing constraints user guide," 2012.

[108] Altera, "Area and timing optimization," 2012.

[109] M. Freeman, "Pcb design conference west," 2000.

[110] D. Stroobandt, J. Depreitere, and J. V. Campenhout, "Generating new benchmark designs using a multi-terminal net model," *Integration, the VLSI Journal*, vol. 27, pp. 113–129, July 1999. ACM ID : 326098.

[111] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, "On routability prediction for field-programmable gate arrays," in *Proceedings of the 30th international Design Automation Conference*, DAC '93, (New York, NY, USA), pp. 326–330, ACM, 1993.

[112] P. Zarkesh-Ha, J. A. Davis, W. Loh, and J. D. Meindl, "Prediction of interconnect fan-out distribution using rent's rule," in *Proceedings of the 2000 international workshop on System-level interconnect prediction*, SLIP '00, (New York, NY, USA), pp. 107–112, ACM, 2000. ACM ID : 333040.

[113] T. Wan and M. Chrzanowska-Jeske, "Generating random benchmark circuits for floorplanning," in *Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on*, vol. 5, pp. 345–348, Sept. 2004.

[114] J. Pistorius, E. Legai, and M. Minoux, "PartGen : a generator of very large circuits to benchmark the partitioning of FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, no. 11, pp. 1314–1321, 2000.

[115] P. Verplaetse, D. Stroobandt, and J. Van Campenhout, "A stochastic model for the interconnection topology of digital circuits," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 9, no. 6, pp. 938–942, 2001.

[116] E. Kuh and T. Ohtsuki, "Recent advances in VLSI layout," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 237–263, 1990.

[117] J. Westra and P. Groeneveld, "Is probabilistic congestion estimation worthwhile ?," in *Proceedings of the 2005 international workshop on System level interconnect prediction*, pp. 99–106, ACM, 2005.

[118] M. Servit, "Prerouting analysis of printed circuit boards," *Computer-Aided Design*, vol. 13, pp. 367–375, Nov. 1981.

[119] OpenMoko.org, “GTA02 outline footprints,”